Tile-based modeling and its applications in computer graphics

Lai, Chi-Fu

2016

Lai, C.-F. (2016). Tile-based modeling and its applications in computer graphics. Doctoral thesis, Nanyang Technological University, Singapore.

<https://hdl.handle.net/10356/69077>

<https://doi.org/10.32657/10356/69077>

NANYANG
TECHNOLOGICAL
UNIVERSITY

Tile-based Modeling and Its Applications in Computer Graphics

LAI Chi-Fu

School of Computer Engineering

2016

Tile-based Modeling and Its Applications in Computer Graphics

LAI Chi-Fu

School of Computer Engineering

A thesis submitted to the Nanyang Technological University
in partial fulfilment of the requirement for the degree of
Doctor of Philosophy

2016

Abstract

Our ancient ancestors invented tiles predominantly for the purpose of decorating architectural interior and exterior structures varying from ceilings, floors to walls. Due to the beauty of tiles, mathematical analysis of 2D tilings has been well studied and classified, for example Wang and Corner Tiles, Periodic Tilings, and Non-Periodic Tilings. Graphics researchers, on the other hand, focused on the discovery of technique on applying tilings to various Computer Graphics classic applications like texture mapping, texture synthesis, and sampling. Applications on surface modeling were freshly proposed at the time this work began. Given the various properties of different types of tiles, say aperiodic, it is well concluded that tile-based techniques often offer a minimization of computational power and memory consumption, while keeping a high degree of freedom to provide wide range of flexible solutions to those classic applications. On the contrary, tiling in 3D space is a relatively new research area, and its applications have great potential in Computer Graphics and Computational Fabrication, which many of them have not yet been discovered until this work or similar works are established.

Furthermore, there are many traditional art works hand-created by making use of 2D tiling concept, for example M. C. Escher's artistic tessellation. Some of those art works present an optical illusion that confuses our visual system in reconstructing 3D geometry from the 2D drawing. There are very limited analysis and research on 3D visualization of such perceptual tiling art works.

This thesis aims to study three various novel forms of tile-based models and their applications: **Tile-based 3D Reconstruction and Navigation of Impossible World**. In this work, we present a tile-based approach towards 3D gaming with impossible figures, delivering for the first time free view navigation in 3D mazes constructed from impossible figures. Such result cannot

be achieved by previous research work in modeling and rendering impossible figures. To deliver seamless gaming navigation and interaction, we propose i) a tile-based 3D reconstruction method that converts the line-art impossible figure into 3D geometry representation, ii) a set of guiding principles for bringing out subtle perceptions and iii) a novel computational approach to construct 3D structures from impossible figure images and then to dynamically construct the impossible-figure maze subjected to user’s view. In the end, we demonstrate and discuss our method with a variety of generic maze types with extended applications.

Tile-based Reconstruction of Surfaces. We introduce a method for optimizing the tiles of a quad-mesh. Given a quad-based surface, the goal is to generate a set of K quads whose instances can produce a tiled surface that approximates the input surface. A solution to the problem is a K -set tilable surface, which can lead to an effective cost reduction in the physical construction of the given surface. Rather than molding lots of different building blocks, a K -set tilable surface requires the construction of K prefabricated components only. To realize the K -set tilable surface, we use a cluster-optimize approach. First, we iteratively cluster and analyze the edge connectivity of the K quads. Then, we apply a non-linear optimization model with constraints that maintain the K quads connections and shapes. Our algorithm is demonstrated on various surfaces, including some that mimic the exteriors of certain renowned building landmarks.

Tile-based Reconstruction of Interlocking Structure. A 3D burr puzzle is a 3D model that consists of interlocking pieces with a single-key property. The intriguing property of the assembled burr puzzle is that it is stable, perfectly interlocked, without glue or screws, etc, so that it is capable to be applied to solve the model partitioning problem of small volume fabrication like home-scale 3D printing. In this work, we generalize the 6-piece orthogonal burr puzzle (a knot) to design and model complex burr puzzles from 3D models. Given a 3D input model, we first interactively embed a tiled network of knots into the 3D shape. Our method automatically optimizes and arranges the orientation of each knot, and modifies pieces of adjacent knots with an appropriate connection type. Then, the entire 3D model is partitioned by splitting the solid while respecting the assembly motion of embedded pieces. Lastly, we also present an automated approach to generate the visualizations of the puzzle assembly process.

Acknowledgments

First of all, I would like to express my appreciation and thanks to my advisor Associate Professor Dr. Chi-Wing Fu. I would like to thank him for encouraging, supporting, and supervising my research. His advices on research have been priceless. I would also like to thank the external thesis examination panel, and the internal thesis examination panel: Dr. Jianfei Cai, Dr. Jianmin Zheng, and Dr. Kemaο Qian for taking their valuable time to review and give constructive comments on my thesis. I would especially like to thank all of my research collaborators, Associate Professor Dr. Ying He, Professor Dr. Daniel Cohen-Or from Tel Aviv University, Professor Dr. Tien-Tsin Wong from Chinese University of Hong Kong, Professor Dr. Chi-Keung Tang from Hong Kong University of Science and Technology, Assistant Professor Dr. Sai-Kit Yeung from Singapore University of Technology and Design, Dr. Shi-Qing Xin from Ningbo University, and Dr. Xiaoqi Yan. Without all their helps, my research and paper publications would not even be possible, neither would this thesis. I would also like to thank Professor Dr. Chi-Keung Tang, Professor Dr. Nadia Magnenat Thalmann from the Institute for Media Innovation, and Assistant Professor Dr. Sai-Kit Yeung for the financial support during my last three years of Ph.D. study. Special thanks to Assistant Professor Dr. Lap-Fai Yu from University of Massachusetts Boston, Dr. Boxin Shi and Dr. Shasha Mao from Singapore University of Technology and Design for their kind support, research discussion and advise.

I also need to thanks all the anonymous participants who helped me to conduct the user study; ustwo studio Ltd. and Goo-Shun Wang for the permission to use the images in Figure 3.1(c), Figure 4.1 and 4.2, respectively. Figures 1.1, 1.2, 1.3, 1.4, 1.5, 3.2, 3.3, 3.4, 3.5, 3.6, 3.13(a), and 6.1 belong to their respective owners. Note that by Copyright Disclaimer Under Section 12.1.11 of the Singapore Commercial Law, allowance is made for “fair dealings” which explicitly applies to use of copyrighted work for research purposes.

In addition, many thanks to all my friends in Singapore, including Dr. Qian Sun, Dr. Min Meng, Dr. Xiaoqun Wu, Ms. Anran Wang, Dr. Long Zhang, Mr. Xiaoning Wang, Dr. Minqi Zhang, Dr. Xiang Ying, Dr. Dayong Wang, Dr. Xiaopei Liu, Mr. Fuwen Tan, Ms. Mengyao Zhao, Mr. Tian Feng and Mr. Zhipeng Mo; and all my friends in Hong Kong, including Ms. Chi-Kiu Lo, Ms. Denise Lee, Ms. Rovenna Chu, Mr. Warner Hong, Mr. Gary Tang, Ms. Ally Leung, Ms. Candy Ng, Mr. Tim Wong, Mr. Maurice Leung, Ms. Maggie Lau, Mr. Jimmy Ko, Mr. Johnson Au, Mr. Timothy Tsang, Ms. Jessica Ng, Mr. Albert Hui and Ms. Daeus KY Tang for their mental supports. I would also like to thank Mr. Tsang Swan Marcus Leo for helping to proofread this thesis.

Lastly but most importantly, million thanks to my family for their understanding on my capricious decision for pursuing a Ph.D. degree.

Contents

Abstract	ii
Acknowledgments	iv
List of Figures	x
List of Tables	xvii
1 Introduction	1
1.1 Background	1
1.1.1 Reviewing Mathematical Interpretation of Tiling	2
1.1.2 Artistic 2D Tessellation	3
1.1.3 Modern Architectural Desire on 2D Tiling	4
1.1.4 Applying Space Filling Concept in 3D Space	5
1.2 Research Objectives	6
1.3 Research Problems	8
1.4 Research Contributions	9
1.5 Thesis Organization	11
2 Related Work	12
2.1 2D Tilings	12
2.1.1 Wang Tiles for Computer Graphics Applications	12
2.1.2 Tilings in Computer Graphics	13
2.2 3D Tiling	14
2.3 Impossible Figures Modeling and Rendering	14
2.4 Architectural Geometry Modeling	15
2.5 Assembly Modeling	16

3	Tile-based 3D Reconstruction of Impossible World	17
3.1	Introduction	17
3.2	Related Works	22
3.3	Constructing Box-based Solid 3D Tiles	25
3.3.1	Inputs and Pre-processing	25
3.3.2	Constructing Box-based Solid 3D Primitives	27
3.3.3	Limitations	32
3.4	Summary	33
4	3D Navigation on Impossible Figures via Dynamically Reconfigurable Modeling	34
4.1	Introduction	34
4.1.1	Related Works	36
4.2	Guiding Principles	39
4.2.1	Conventional 3D animations	39
4.2.2	Guiding principles	40
4.3	Dynamically Reconfigurable Modeling	41
4.4	Adaptation to Different Scene Types	45
4.4.1	Achieving Seamless Viewing	46
4.4.2	Different Generic Scene Type	46
4.5	Adaptation to Other Application	53
4.5.1	Haptics Interaction Scenario	53
4.5.2	Virtual Reality Experience	57
4.6	User Study	57
4.7	Limitations	62
4.8	Summary	63
5	Tile-based Reconstruction of Surfaces	65
5.1	Introduction	65
5.2	Overview	66
5.3	Related Works	68
5.3.1	Research Works on Discrete Differential Geometry	68
5.3.2	Concurrent Similar Research Works	69

5.4	Clustering	69
5.4.1	Initial Clustering	70
5.4.2	Cluster Merge	72
5.5	Optimization	73
5.6	Implementation Details and Results	76
5.6.1	Clustering Enhancement	76
5.6.2	Results	78
5.6.3	Limitations	80
5.7	Summary	81
6	Tile-based Reconstruction of Interlocking Structure	82
6.1	Introduction	82
6.1.1	Background on Burr Puzzles	84
6.1.2	Properties on Burr Puzzles	85
6.2	Overview	86
6.3	Single-Knot Burr Puzzle	87
6.3.1	Locking Mechanics	88
6.3.2	Knot Embedding	88
6.3.3	Flesh Attachment.	90
6.4	Multi-Knot Burr Puzzle	91
6.4.1	Knot Network Design	92
6.4.2	Knot Orientation	93
6.4.3	Knot Connection and Dependency	94
6.4.4	Determining Knot Orientations and Connections	95
6.4.5	Generating Puzzle Pieces	98
6.4.6	Extensions	98
6.5	Assembly/Disassembly Illustration	100
6.5.1	Camera Movement	101
6.5.2	Motion Arrows and Numbering	101
6.5.3	NPR Stylization and Color-Coding	102
6.6	Result	102

6.6.1	Rapid-prototyping Results.	105
6.7	Implementation Issues and Limitations	105
6.7.1	Implementation Issues.	105
6.7.2	Limitations.	106
6.8	Summary	107
7	Conclusion and Future Work	108
7.1	Conclusion	108
7.2	Future Work	110
	References	112
	Publications	121

List of Figures

1.1	A tiled floor of a church in Seville, Spain, that uses three different kind of tiles in shape of square, triangle and hexagon, respectively.	1
1.2	The three only existing regular tilings: (from left to right) the equilateral triangle, square, and regular hexagon.	2
1.3	(a) A color-coded parallelogram tessellation that gives spatial feeling of the 2D projection of infinite number of 3D cubes; and (b) an impossible figure constructed by the set of impossible tiles discovered by Diego Uribe [86]. . . .	3
1.4	Buildings with curved exteriors: Swiss Re Tower in London, De Beers Ginza in Japan, and Turning Torso in Sweden.	4
1.5	An example of 3D tiling, where all the tiles are color-coded solid 3D model. . .	5
3.1	(a) Ascending & Descending, M. C. Escher's painting showing monks walking indefinitely in the INFINITE STAIRCASE; (b) a making-of photograph showing the INFINITE STAIRCASE in the movie <i>Inception</i> ; and (c) a game level in <i>Monument Valley</i> , embedding an impossible-figure structure.	18
3.2	Classes of impossible figures: i) <i>depth interposition</i> , e.g., IMPOSSIBLE CUBOID (a); ii) <i>disappearing normals</i> , e.g., IMPOSSIBLE STAIRCASE (b); iii) <i>disappearing space</i> , e.g., IMPOSSIBLE TRIDENT (c); and iv) <i>depth contradiction</i> , e.g., PENROSE TRIANGLE.	18
3.3	(a) PENROSE TRIANGLE photographed at a specific view of (b) a highly-twisted solid; (c) another PENROSE TRIANGLE based on (d) a disconnected paper model.	19

3.4	(a) The set of tiles proposed by Diego Uribe [86] for constructing impossible - or possible - structures made of square bars meeting at right angles; and (b) an example impossible figure constructed using the set of tiles that mimics the middle portion of middle Escher's <i>Belvedere</i>	20
3.5	(a) Another set of tiles proposed by Diego Uribe [86] for constructing impossible (possible) structures that contains two different views, so that parts of the figure are seen from above while other parts are seen from below; and (b) an example impossible figure constructed using the set of tiles shown in left. . . .	21
3.6	(a) A joint (possible object) centred in an undefined plane-tiling polygon exists in six versions, one for each orientation; and (b) a catalogue of all available joints in which bars abut on a point. Taking account into orientation, there are 64 different joints in total.	21
3.7	Our algorithm (a) takes an impossible figure (RGB image and normal map) as input, (b) converts it into a set of triangles or primitives where the geometric relationship are encoded by a relative position graph, and then (later in Chapter 4) (c) adapts it for different applications. Note that the face normal of the triangles in (b) is retrieved directly from the normal map, for illustration consistence, we use the same color coding from the normal map for the normal arrows. Please refer to Section 3.3.1 for a detail explanation on the relative position graph.	24
3.8	(a & b): our simple rotating-cube program for finding normal map colors of impossible figure images. (c) we then use GIMP to fill corresponding regions in the figure image.	25
3.9	Partition the foreground region of the input impossible figure into quadrangles: (a) an input impossible figure, and the three major axes in orthogonal projection; (b) extract contour lines and identify their directions; (c-e) extend the contour lines one by one in the image space to partition the figure, see (e) for the result and Algorithm 1 for the pseudo code outline.	26
3.10	(a) Occluded linkage: the dotted line between the two visible blue lines; (b) Star-junction (orange dots), where six contour lines meet and a cube is hidden behind.	28

3.11	(a) A single box. (b) orange circles mark partially-occluded faces (partial quadrangles) of boxes. (c) box-fitting results: we assign the same color to the faces of the same box.	30
3.12	Three different groups of patterns for box-fitting; note that these patterns can be rotated during the matching.	31
3.13	(a) An input impossible figure with a covered region behind the wall (see arrow); there could be many possible ways of reconstruction; (b) extra user input is needed, if we want to reconstruct the hidden structure marked by the arrow; otherwise, our current method will leave it unconnected.	33
4.1	A key frame sequence (following the orange arrows) in 3D animation <i>Hallucii</i> , showing a drunk man who walks over a 3D infinite staircase. The first image in the sequence shows a local view on the drunk man while the others show the entire staircase.	38
4.2	Another key frame sequence from <i>Hallucii</i> , showing that the drunk man finds the <i>same</i> briefcase again at the <i>same</i> corner after completing one cycle down the 3D infinite staircase. The yellow arrows indicate his walking direction. . . .	38
4.3	2D visualization of the dynamically reconfigurable model generated from Algorithm 2 using the image-based triangles representation of the PENROSE TRIANGLE. The blue dots in (a-d) indicate the avatar location (P_c); the grayscale shading of the triangles indicate the screen-space z value, where the darker the shading, the smaller the z value. Note that the gaps are located at the discontinuity of the grayscale shading as shown in (a) as an example.	44
4.4	Endless runner game prototype: (a) the input impossible figure (top) and the maze model dynamically reconstructed by Algorithm 2 w.r.t. P_c (green dot); (b&c) the starting and ending views, respectively, where the game avatar keeps running upwards endlessly to collect the coins on the impossible-figure maze; (d) without the adaptation strategy, the gap can be seen when the game player looks around; and (e) the adaptation strategy can help avoid the gap and achieve <i>seamless viewing</i>	44

4.5	Chasing game prototype: (a) input impossible figure (box-based solid primitives); (b) the maze model built from Algorithm 2 w.r.t. P_c (red box); and (c-e) subsequent screenshots, demonstrating <i>subtle impossibility perception</i> when we move over the maze, see also the mini-map (lower-left).	47
4.6	(a,b) our box fitting method can also work with impossible figures of the depth interposition class. (c) by manually connecting the interposition links with invisible boxes (see the double-ended arrows in (a,b)), our method can also work with this class of impossible figure.	48
4.7	Screenshots from the game prototype (Role-Playing Game) that mimics the Arcane Sanctuary level in Diablo II (which is simply a 2.5-D impossible world). <i>Zoom-in</i> : an example position and view direction of the avatar. (a-c): our method can reconstruct this impossible-figure world in 3D.	50
4.8	Multi-player shooting game prototype: The nontrivial impossible figure (a) from which the maze of corridors is built; and the maze model (d) reconfigured by Algorithm 2 w.r.t. P_c (the red box); Player 1 (in blue) took the lift to move up to a seemingly “upper” level (b). Next, Player 2 (in brown) entered the game and found Player 1 inside the lift above him (e). After adventuring through the impossible-figure maze for some time, the two players encountered each other in their own views, and then try to shoot at each other (c,f).	51
4.9	An exemplary interaction scenario of using the haptic device to touch and feel an impossible figure PENROSE, whose structure was used to build the infinite staircase in <i>Inception</i> . Note that the 3D model (shown in the bottom) is broken most of the time (but not sensible by the users), as indicated by the abrupt depth discontinuities.	54
4.10	A flattened figure resembling the impossible figure input in the multi-player shooting game prototype.	58
4.11	Statistical results gathered from the twenty participants who tried our gaming prototype: mean and standard deviation of variables (a1)-(a5) and (b1)-(b3) for the case of impossible-figure maze (left) and possible 3D maze (right).	61

5.1	Overview of our approach over a MONKEY SADDLE surface: We iteratively cluster and analyze: For this example we (i) cluster the quads into 24 clusters, and (ii) analyze the edge connections to detect 33 distinct edge lengths in the specific clustering configuration. Then (iii) we optimize the given surface into a KT-surface.	66
5.2	(a) original surface; (b) simply replacing surface quads with S-quads; (c) optimized S-quads tiled on KT-surface.	67
5.3	Affinity matrices (linearly color-mapped) for (a) MONKEY SADDLE and (b) SEASHELL surfaces.	70
5.4	Given four clusters of quads on MONKEY SADDLE (a), we analyze the edge sharing conditions for S-quad instances by an edge sharing graph (b). Edges marked with the same color should have matched edge length on the KT-surface (c).	71
5.5	(a) K-set DE BEERS GINZA ($K = 27$); (b) original surface of SWISS RE TOWER; (c) K-set SWISS RE TOWER ($K = 21$). There is only a little visual degradation for smaller K	78
5.6	Resultant K-set tilable surfaces (from top to bottom): SWISS RE TOWER (528 quads); TUNNEL (1152 quads); SEASHELL (400 quads); DECOCUBE (480 quads); MONKEY SADDLE (100 quads); and DE BEERS GINZA (80 quads).	79
5.7	The non-uniform arrangement of quads around the TOWER tip.	80
6.1	Different kinds of 3D interlocking puzzles that resemble different objects.	82
6.2	(a) A burr puzzle made from BIMBA; and (b) the eleven puzzle pieces after disassembly.	83
6.3	(a) A traditional cuboid burr puzzle; and (b) the canonical six-piece burr puzzle.	84
6.4	(a) Single knot: the canonical six-piece burr; (b) embedding the single knot into the 3D model, SQUIRREL; (c) extending the “blade” to partition the volume without touching the center burr lock; and (d) pieces after flesh attachment.	86
6.5	Possible disassembling orders: the vertical rectangular boxes and rounded squares next to each of them denote the states and the removable puzzle piece(s) at each state, respectively.	89

6.6	(a) Two-way attachment; and (b) even attachment by three 45-degree cutting planes.	90
6.7	Fragments may be resulted during partitioning.	91
6.8	Constructing a multi-knot burr puzzle: (a) a given 3D model; (b) a network of knots; (c) partition the model volume for each knot; (d) compute the knot interdependency; (e) compute the knot orientation; and (f) generate the puzzle pieces.	92
6.9	Connection types of neighboring knots.	93
6.10	Illustrating knot orientations and connections determination using the ISIDORE HORSE as an example: <i>Step 1: Compute a shortest path tree</i> : (a) Note that red circled knot indicates the user specified key K and is the root node of the shortest path tree, and blue lines indicates the edges of the tree. <i>Step 2: Candidate knot orientations</i> : (b) K is rotated to face to the back of the ISIDORE HORSE, so that K ₁ does not face to any neighbouring knots; (c) other knots are rotated so that their key pieces are facing to their corresponding parent knots. <i>Step 3: Candidate connection types</i> : (d) Type #1 is being chosen for each pair of neighboring knots. <i>Step 4: Determining orientation and connection</i> : after simulating the disassembly starting from K , all pieces can be disassembled from the initial orientation and connection. Note that all B ₃ and B ₅ are facing away from neighboring knots.	96
6.11	(a) Non-orthogonal connection; and (b) connecting disjoint knot networks. . . .	99
6.12	Illustrating the puzzle disassembly: (a) motion arrow placement to indicate puzzle piece movement; (b) number label to show the movement stage; (c) contact areas coding with zoom-in view to illustrate the interlocking structure; (d) successive views showing the camera movement to expose the next puzzle piece in-motion; and (e) annotating multiple puzzle pieces in motion.	100
6.13	Knot networks employed to construct the 3D burr puzzles presented in Figure 6.14.	103
6.14	Single-knot and multi-knot burr puzzles created from assorted 3D models: (a) CUBICBOX, (b) BUNNY, (c) 2-TORUS, (d) MOAI, (e) DRAGON, (f) LAURANA, (g) SHARK, (h) TORUS, (i) SQUIRREL, (j) MEGABOX, (k) ISIDORE HORSE, (l) BIMBA, and (m) CHESS.	104

6.15	Rapid prototyping models of our 3D burr puzzles.	106
6.16	An assembly sequence of the BUNNY puzzle.	107

List of Tables

4.1	Notations and functions used in Algorithm 2.	41
5.1	Optimization results on surfaces shown in Figure 5.6.	77
6.1	Information about the burr puzzles in Figure 6.14.	104

Chapter 1

Introduction

1.1 Background

Tiling is traditionally used on the planar surfaces of ancient buildings like floors, walls, and ceilings for decorative purpose formed by patterns of clay tiles such that the tiled surface is fully covered with no overlaps and gaps between tiles [46]. Figure 1.1 shows an example floor tiled by three unique tiles of different shapes and colors, forming a nicely designed floor. In science and engineering, tiling in 2D space is a geometry topic that studies how an unique set

Figure 1.1: A tiled floor of a church in Seville, Spain, that uses three different kind of tiles in shape of square, triangle and hexagon, respectively.

Figure 1.2: The three only existing regular tilings: (from left to right) the equilateral triangle, square, and regular hexagon.

of shapes, or tiles, can be arranged (under a set of preset rules) to fill a 2D surface without gaps and overlaps between tiles.

1.1.1 Reviewing Mathematical Interpretation of Tiling

Mathematically, a tiling (or tessellation) is a coverage of the 2D Euclidean plane by a countable number of closed sets, called tiles, such that the tiles must intersect only on their boundaries. The segment of line or curve formed at the intersection of two bordering tiles is defined as an edge; for tiles with regular shapes, an edge is always a straight line [30]. The point of intersection of three or more bordering tiles is defined as a vertex. The duplicating region that forms the tessellation is called the fundamental region. There are only three existing regular tessellations (see Figure 1.2): the equilateral triangle, square, and regular hexagon, where the tessellation has identical vertices, and the angle between the adjacent edges of every tiles are the same.

Figure 1.3: (a) A color-coded parallelogram tessellation that gives spatial feeling of the 2D projection of infinite number of 3D cubes; and (b) an impossible figure constructed by the set of impossible tiles discovered by Diego Uribe [86].

1.1.2 Artistic 2D Tessellation

Tessellation has existed for a long time as pure artistic painting. M. C. Escher [56] is one of the well-know artists who has significant amounts of tessellation painting collections. The mechanism of his tessellation works are well studied and analyzed [76]. In computer graphics, researchers like Kaplan and Salesin have proposed computational method [37, 38] for Escherization - reproducing M. C. Escher's tessellation.

Other than artworks that directly produces tessellation, M. C. Escher is strong in applying modularity [36] to produce his artworks. The concept of modularity in Art is to use a limited finite set of basic element (modules) for constructing a large collection of various possible (modular) structures, where some of them can provide strong spatial feeling to the human visual system. For example, if we color code a parallelogram tiling as shown in Figure 1.3 (a), it gives a feeling of seeing a projection of infinite number of 3D cubes. One classic example is the impossible figures (e.g. Figure 1.3 (b)) that essentially confuse our visual system to subconsciously interpret the 2D drawing as a projection of a 3D object that is structurally

Figure 1.4: Buildings with curved exteriors: Swiss Re Tower in London, De Beers Ginza in Japan, and Turning Torso in Sweden.

inconsistent to any real-world object.

1.1.3 Modern Architectural Desire on 2D Tiling

Prefabrication is standard practice nowadays in architectural building construction, where prefabricated components, or semi-finished parts, are used to assemble walls, panels, and ceilings [5]. These prefabricated components are pre-built offsite, transported to the construction site, and further assembled onsite. The use of prefabricated components has a number of advantages, such as reducing construction time. Its applicability to modern curved building exteriors (like those shown in Figure 1.4) is, however, limited by the distinct panel shapes that curved surfaces require. Undoubtedly, if we apply basic symmetries, each prefabricated panel has a unique shape on the curved surface. Unlike ancient architecture that focuses on applying tiling for decoration purpose, however, in modern architecture, it is not hard to link the prefabrication process to the concept of tiling, where we can consider the set of mold for producing the prefabricated components to be the set of tile.

Figure 1.5: An example of 3D tiling, where all the tiles are color-coded solid 3D model.

1.1.4 Applying Space Filling Concept in 3D Space

Though it can be rarely found in real-life, mathematically, tiling can be extended from 2D to other dimension in Euclidean space. The main spirit of tiling is a partition of space into tiles that are topological cells, i.e. bounded bodies that have no holes existed. A real-life example of non-periodic 3D tiling is dry foams; Figure 1.5 shows an illustrating virtual example of 3D tiling.

Sharing the same spirit with the prefabrication in architectural surface stated in the previous section, tiling in 3D space can be applied in solid model fabrication to reduce the construction time. Different from architectural usage, if we want to fabricate solid model for other purposes, e.g. artwork, we may not want to use glue, screws or nails to assemble the components, rather have them interlocking to each other. Thus, it is important to study on a method that can i) partition a 3D space (model) into smaller components, and ii) the components are self-interlocking post assembly.

1.2 Research Objectives

Echoing Section 1.1, we identified three major research problems that are unsolved before this work is done:

- (i) **Novel 3D modeling and visualization method for tile-based impossible figure (as elaborated in Section 1.1.2).** Previous research on impossible figures focuses extensively on single view modeling and rendering. Existing computer games that employ impossible figures as navigation maze for gaming either use a fixed third-person view with axonometric projection to retain the figure’s impossibility perception, or simply break the figure’s impossibility upon view changes. Visualizing impossible figure in 3D space is challenging as there are no physically existing objects that can completely align with how our visual system reconstructs the object presented in a 2D impossible figure. Thus, it is difficult for normal people to perceive the impossibility of the impossible figure compared to a normal physical object. We have to develop a novel 3D modeling method that can recreate the geometry of the perceived objects, and a novel visualization method that visualize such a geometry representation while keeping the impossibility perception unchanged.
- (ii) **Optimization on the balance between construction cost and surface detail on architectural surfaces (as elaborated in Section 1.1.3).** In modern architecture, prefabricated components, or semi-finished parts, are used to assemble building exteriors. As the architectural exterior of a building becomes increasingly complex, the number of distinct prefabricated components undoubtedly increases, and ultimately so will the construction cost. It is non-trivial to find a balance between minimizing the number of distinct prefabricated components whilst keeping the building’s exterior surface detail unchanged as much as possible. We have to derive a modeling scheme with optimization involved to

preserving the surface detail while we lower the number of distinct prefabricated components.

(iii) **Decomposing a solid model into multiple smaller pieces that can be (dis)assembled stably without the use of glue, screws, and nails (as elaborated in Section 1.1.4).**

Though 3D printing technology is emerging, printing a large scale 3D solid model is still time-consuming. One way to hasten the production time is to divide the solid model into independent smaller pieces, and print them simultaneously. However, the assembling process still requires gluing or adding screws and nails which might cause irreversible damage to the printed components should any mistakes occur. Additionally, gluing and nailing are irreversible processes which will decrease the mobility of the printed model; screwing and nailing will also often affect the aesthetic appearance of the printed model. We seek a more intellectual method that can decompose a solid model into independent smaller pieces that can be assembled, disassembled and reassembled again stably without the need of glue, screws and nails.

Our research objectives can be summarized in one sentence: We aim at investigating different novel forms of 3D models that are generated from 2D tilings or inspired by tiling techniques that can rectify the above-mentioned problems.

In detail, for each of the above stated problems, our solutions should satisfy the following requirements:

- (i) **Requirements of the Research Objective item 1:** The goal of having such a 3D modeling and visualization method is to help the user to understand the impossibility from the impossible figure by exploring such a non-physically existing 3D impossible object in 3D space. We understand from Wu et al. [96] that even applying their work, it is impossible to produce a valid 3D object from an impossible figure for all viewing angles.

Hence, aiming at visualizing the entire regenerated impossible object in a view frustum is extremely challenging. Thus, we only require our 3D modeling and visualization method to provide a 3D navigation experience such that, from the aspect of the user, 1) he/she should not visually notice any structural inconsistencies on the geometry representation with its corresponding impossible figure; 2) the display should always look like an Euclidean geometry to the user, and 3) after careful exploring, the user should be able to experience the same kind of impossibility perceived from the original 2D impossible figure.

- (ii) **Requirements of the Research Objective item 2:** Recaping that the goal of the desired optimized architectural surfaces is to have as little number of prefabricated components as possible while best preserving the surface details. Thus, it is obvious that the optimization framework has to produce results that facilitate the above needs. Additionally, the framework has to be scalable and flexible such that change of optimization goal as well as input requirement can be easily accomplished; the user should be able, but not required to, specify the desired number of prefabricated components, in order to allow manual control for special architectural needs.
- (iii) **Requirements of the Research Objective item 3:** Our solution only needs to deal with closed 2-manifold in 3D space. The solution should be flexible enough to allow the user to have a certain level of control over the size of the resulting decomposed solid model pieces. The model pieces should then be able to be assembled in a way that the completed solid model is self-interlocking. The sequence of assembling should be able to be visualized in a clear way.

1.3 Research Problems

In this section, we will be defining the research problem statements that we solve individually.

- (i) **Tile-based 3D Reconstruction of Impossible World (Chapter 3):** Given a 2D impossible figure image as input, the goal is to generate the geometry representation of the input impossible figure that consists of a set of tiled 3D boxes.
- (ii) **3D Navigation on Impossible Figures via Dynamically Reconfigurable Modeling (Chapter 4):** Given the geometry representation of an 2D impossible figure (from Chapter 3), the goal is to be able to perform 3D navigation on an impossible world that is structurally consistent with the 2D impossible figure.
- (iii) **Tile-based Reconstruction of Surfaces (Chapter 5):** Given a quad-based surface as an input, the goal is to generate a K -set tilable surface (or KT-surface) with a prescribed K while maintaining the KT-surface close to the input surface.
- (iv) **Tile-based Reconstruction of Interlocking Structure (Chapter 6):** Given a solid 3D model, the goal is to tile the 3D sharp space using multiple burr puzzles, and partition it into perfectly interlocking puzzle pieces that can be disassembled with a single movable key piece.

1.4 Research Contributions

This thesis collects four major research contributions that solve and satisfy the problems and requirements stated in Section 1.3 and 1.2, respectively. They are described separately as follows:

- (i) **Tile-based 3D Reconstruction of Impossible World:** We revisited the notion of impossible figures, and then studied the analysis of the set of impossible tiles, and gathered a list of structural properties. Based on results derived from the analysis, we proposed an automatic tile-based 3D reconstruction method that analyzes and turns line-art images of

impossible figures into box-based 3D structures - the geometry representation, that can be used to construct a 3D impossible world.

(ii) **3D Navigation on Impossible Figures via Dynamically Reconfigurable Modeling:**

We present a set of *guiding principles* for producing seamless 3D navigation and interaction experience when visualizing a geometry representation from an impossible figure. We then propose a novel procedure that dynamically reconnects and re-structures an impossible-figure geometry based on the user's view, so that we can achieve the guiding principles and enable seamless single-player navigation and interaction. Lastly, we apply our modeling technique on gaming and propose adaptation and extension for different game maze types: i) *open space*, ii) *a maze of corridors*, and iii) *multiple platforms*. To show the feasibility of our methods, we developed a game prototype for each of them and conducted a user study.

(iii) **Tile-based Reconstruction of Surfaces:** In order to obtain a solution to the architectural problem stated in the previous section, we analyze the surface structure of a quad-based surface mesh, which is commonly used in the production of exterior panels in architecture. The proposed solution has three major research contributions involved. First, we solve the architectural problem by introducing the notion of K-set tilable (KT) surfaces. Then, we present a mathematical framework to convert a quad-based surface mesh to a KT-surface based on non-linear optimization. Lastly, we show the construction of various KT-surfaces that are quantitatively close to the input surfaces.

(iv) **Tile-based Reconstruction of Interlocking Structure:** We derived a solution inspired by traditional burr puzzle to solve the solid model decomposition problem. We analyzed the ability of tiling a 3D space by placing multiple instances of burr puzzle with the same interlocking structure, and generalize a set of rules to combine neighbouring burr puzzles while enforcing the single-key interlocking and (dis)assemble ability of the resulting

puzzle. To help on the (dis)assembly sequence, we also propose a visualization system to automatically illustrate and annotate the puzzle assembly process. We demonstrate the feasibility of the approach by producing 3D printed rapid prototypes from different 3D models.

1.5 Thesis Organization

The remainder of this thesis is organized as follows: Chapter 2 reviews Computer Graphics applications of different types of 2D tilings. We focus more on applications more related to modeling. We then review the state-of-the-art research on 3D tilings. Chapter 3 presents a method to tile-based 3D reconstruction method for generating a geometry representation of a 2D impossible figure. Chapter 4 describes a dynamically reconfigurable modeling procedure to achieve 3D navigation of a tile-based impossible figure by making use of the geometry representation from Chapter 3. Chapter 5 introduces a mathematics notion of K-set tilable surface which simplify a surface quad mesh to form by a set of K unique quad while having its surface details preserved as much as possible. Chapter 6 describes a method that tiles a 3D burr puzzle inside a 3D model, to generate fabricable pieces that are single key interlocking after assembly. Finally, Chapter 7 draws a conclusion and discusses possible future work.

Chapter 2

Related Work

In this Chapter, we review various Computer Graphics applications that utilize techniques based on different types of tiling. In particular, we review applications that based on 2D tilings (Section 2.1), and 3D tilings (Section 2.2). We also review related works for each of our research problems:

- (i) Section 2.3 for Tile-based 3D Reconstruction of Impossible World and 3D Navigation on Impossible Figures via Dynamically Reconfigurable Modeling;
- (ii) Section 2.4 for Tile-based Reconstruction of Surfaces; and
- (iii) Section 2.5 for Tile-based Reconstruction of Interlocking Structure.

2.1 2D Tilings

2.1.1 Wang Tiles for Computer Graphics Applications

In early 1961, mathematician Hao Wang proposed a class of formal systems namely Wang Tiles [91], that are square tiles with color on each side. Wang Tiles are well-known on its aperiodic property without the need to rotate or reflect the tiles, which make them to be one

of the most powerful foundation on generating complex signals for Computer Graphics usage, such as texture or sampling points, which can be hardly synthesized and stored efficiently.

In 1997, Stam [83] first brought Wang Tiles to the Computer Graphics field by introducing a method that generates non-repeating water textures and caustic maps using an aperiodic set of 16 Wang Tiles pieces. Cohen et al. [11] then popularized the use of Wang Tiles by presenting a stochastic system for non-periodically tiling the plane with a small set of Wang Tiles. The same work also demonstrated the application of such a system in 2D texture, 2D Poisson distributions, and 3D geometry synthesis. Fu and Leung [24] extended the texture tiling mechanism of Wang tiles for synthesizing textures on arbitrary topological surfaces. Lagae and Dutré [45] formulated a procedural object distribution function that follows the Poisson disk tiles inspired by Wang Tiles. Kopf et al. [44] used recursive Wang Tiles for rapidly generating large point sets possessing a blue noise Fourier spectrum. Other than Wang Tiles, we also reviewed other tile-based techniques and their applications in the following subsections.

2.1.2 Tilings in Computer Graphics

One type of artwork from M. C. Escher specifically produces periodic tilings using different shapes. Kaplan and Salesin [37] used periodic isohedral tilings to transfer any single 2D shape into the tiling artwork of M. C. Escher and defined the process to be *Escherization*. A few years later, Kaplan and Salesin [38] extended the support to Escherization by producing aperiodic tiling of two different 2D shapes. Hausner [31] designed a system for simulating decorative tile mosaics. Kim and Pellacini [42] introduced Jigsaw Image Mosaic that are arbitrary shape of tiles that compose a picture. Ostromoukhov et al. [66] proposed to use a hierarchically subdivided Penrose tiling to generate well-distributed sampling point that enforced the blue noise properties.

2.2 3D Tiling

Another type of artwork from M. C. Escher involves carving tiling onto a 3D sphere, yet, only a few pieces of this artworks are produced due to the complexity and high time-consumption. Yen and Séquin [100] developed an interactive tool to design and manufacture this kind of “Escher Spheres”. A 3D solid model can be decomposed into a set of 3D tiles simply by identifying the regular structure. Mitra et al. [58, 59] demonstrated that by first detecting partial and approximate symmetry of a 3D geometry and then deform the 3D model to enforce certain symmetry. Van Wijk [89] presented a method for generating space models of regular maps - a tiling of a closed 3D surface into faces, bounded by edges that join pairs of vertices, such that these elements exhibit a maximal symmetry. Schiftner et al. [77] introduced circle/sphere packing meshes with regard to both their geometry and approximation properties.

2.3 Impossible Figures Modeling and Rendering

In this section, we discuss various areas of research work related to impossible figures and 3D dynamic and procedural modeling. Previous works in computer graphics research focused mainly on generating plausible views of an impossible figure. Tsuruno [85] created an interesting computer animation that detailed different views of *Belvedere*, one of M.C. Escher’s drawings [56] with impossible figures. Later, Savransky et al. [75] proposed using relative transformations from user input to model impossible figures. Khoh and Kovesi [39] first developed a method to model impossible figures, so that we can create a perception of rotating an impossible figure. However, their method is only applicable to certain impossible figures. Later, Owada and Fujiki [68] developed a more general optimization method to broaden the range of plausible views, and built a game-like system called theRelativity [25, 26]. However, the range of viewing angles on impossible figures is still limited. Hence, Wu et al. [96]

proposed using user-markup constraints to model and render impossible figures through a thin-plate-spline optimization model, so that much wider plausible viewing ranges can be achieved at interactive speed. More recently, Elber [21] proposed using line-of-sight deformation to physically construct impossible figures at pre-conditioned views. However, the resulting geometric models of impossible figures produced from Wu et al. [96] and Elber [21] are highly twisted in the world space, so we are unable to apply these methods to model a navigable virtual world as in our case. Compared to the above works, our work models impossible figure utilizing a graph representation that can be applied to 3D navigation driven by a novel approach. Such result cannot be achieved by any of the above computer graphics methods for modeling impossible figures.

2.4 Architectural Geometry Modeling

Architectural geometry is one computer-aided-geometric design area emerged at the borders between discrete differential geometry and architectural engineering. This area is primarily driven by the increasing demands in designing and modeling freeform surfaces. Motivated by practical architectural needs, Liu et al. [51] introduced conical meshes, which have planar faces and yet possess offset meshes at constant face-to-face distance from the base mesh. They developed an optimization model to convert quad-meshes into canonical meshes and combined Catmull-Clark subdivision with their approach. Yan et al. [98] proposed a variational approach to extract general quadric surfaces from mesh surfaces; quadric proxies are progressively inserted (or merged) against an error threshold to improve the surface approximation. Building upon the concept of parallel meshes, Pottmann et al. [71] developed methods to optimize meshes with offset properties relevant to architectural modeling. Wang et al. [92] further generalized the quadrilateral conical meshes to hexagonal meshes and revealed the natural correspondence between regular triangular meshes and P-Hex meshes. By Dupin duality, they

further developed an optimization framework to convert freeform surfaces to P-Hex meshes. Later, Pottmann et al. [72] systematically investigated semi-discrete surfaces and presented algorithms to optimize freeform surfaces into developable strips, leading to elegant solutions for surface panelization. Kilian et al. [40] analyzed developable surfaces with curved creases and applied them to assorted architectural and industrial designs. More recently, Schiftner et al. [77] introduced circle-packing meshes, where the incircles on neighboring triangle faces touch one another and the induced spheres centered at each vertex also form a packing. We propose an optimization framework that minimizes the number of prefabricated components required to build a panel surface.

2.5 Assembly Modeling

Graphics researchers proposed techniques for *recreational assembly*, including the construction of papercraft toy models by Mitani and Suzuki [57], the plush toys by Mori and Igarashi [62], the digital bas-relief models by Weyrich et al. [93], the paper-folding models by Kilian et al. [41], the polyomino 3D puzzle models by Lo et al. [52], the shadow-art models by Mitra and Pauly [60], and the paper-popup architectural models by Li et al. [48].

Since the 60's, computational methods have been developed for solving puzzles; and in particular, 2D jigsaw puzzles. Freeman and Garder [23] were pioneers approaching this problem by considering the geometry of puzzle pieces. Wolfson et al. [94] introduced a two-stage puzzle assembly algorithm. Goldberg et al. [29] further developed techniques to improve the performance and robustness. Rather than apictorial, Cho et al. [9] recently developed a probabilistic approach for solving 2D image jigsaw puzzles. While prior works focused mostly on solving 2D jigsaw puzzles, our method creates a practical approach by interlocking 3D burr puzzles from general 3D shapes. Without the aid of a computer, such interlocking 3D puzzles are extremely difficult to design, and can only be developed by highly skilled craftsmen.

Chapter 3

Tile-based 3D Reconstruction of Impossible World

3.1 Introduction

The first impossible figure was believed to be designed in 1934 by the Swedish artist Reutersvard, who found a new way of arranging nine cubes in a drawing [49]. In the later years, M. C. Escher [56] popularized impossible figures and ingeniously embedded them into a number of his famous drawings; one example is a famous building painting named “Ascending & Descending” (Figure 3.1(a)), where an INFINITE STAIRCASE is embedded at the roof level of the building with two queues of monks walking on it infinitely in ascending and descending order, respectively.

The Notion of Impossible Figure. An impossible figure is a type of optical illusion that confuses our visual system in reconstructing 3D geometry from the 2D drawing. In general, there are four classes of impossible figures (also introduced in Ernst [22], and Wu et al. [96]), each with a different construction mechanism for achieving the perceptual impossibility:

- *Depth contradiction*, where the propagation of local 3D perception information produces a global structural inconsistency over the figure. The INFINITE STAIRCASE (Figure 3.1(a&b)) and the PENROSE TRIANGLE (Figure 3.3) both belong to this class.

Figure 3.1: (a) Ascending & Descending, M. C. Escher's painting showing monks walking indefinitely in the INFINITE STAIRCASE; (b) a making-of photograph showing the INFINITE STAIRCASE in the movie *Inception*; and (c) a game level in *Monument Valley*, embedding an impossible-figure structure.

Figure 3.2: Classes of impossible figures: i) *depth interposition*, e.g., IMPOSSIBLE CUBOID (a); ii) *disappearing normals*, e.g., IMPOSSIBLE STAIRCASE (b); iii) *disappearing space*, e.g., IMPOSSIBLE TRIDENT (c); and iv) *depth contradiction*, e.g., PENROSE TRIANGLE.

- *Depth interposition*, as represented by the IMPOSSIBLE CUBOID (Figure 3.2(a)), where the visual illusion is due to the deceitful depth ordering between object parts; and
- *Disappearing normals*, as represented by the IMPOSSIBLE STAIRCASE (see Figure 3.2(b), not to be confused with the infinite staircase), where the illusion originates from a problematic plane, that appears to be horizontal or vertical depending on where it is connected and seen.

Figure 3.3: (a) PENROSE TRIANGLE photographed at a specific view of (b) a highly-twisted solid; (c) another PENROSE TRIANGLE based on (d) a disconnected paper model.

- *Disappearing space* has an incomplete silhouette that results in a unresolvable foreground and background; a classic impossible figure of this class is the IMPOSSIBLE TRIDENT (Figure 3.2(c)).

Since our visual system subconsciously tries to interpret a 2D drawing as a projection of 3D objects, it will attempt to reconstruct an impossible figure's 3D geometry when such a figure is given. Therefore, when we start to look at *a small local region* on the figure, the interpretation works well: locally as a normal (possible) 3D object. However, problems could arise in the interpretation when we consider *a larger region* on the figure, since certain geometric contradictions could turn out when our visual system attempts to interpret the figure's full 3D geometry. This happens for all classes of impossible figures, see again Figure 3.2.

Photographers create impossible figure photo-shooting by making use of the fact that any photograph is a projection of 3D object(s) onto a 2D plane. Given a target impossible figure photograph, there are infinite number of 3D object projection solution to yield the same result. For example, making use of the fact that by adjusting the camera view, the projection of some curved edges are straightened. Thus, a highly-twisted solid object (like Figure 3.3 (b)) can be photo-shot as an impossible figure photograph. Another example is making use of the fact that the projection of the edges of a 3D object can be overlapping on other edges of its own, and

Figure 3.4: (a) The set of tiles proposed by Diego Uribe [86] for constructing impossible - or possible - structures made of square bars meeting at right angles; and (b) an example impossible figure constructed using the set of tiles that mimics the middle portion of middle Escher’s *Belvedere*.

thus, photo-shooting a disconnected object (like Figure 3.3 (d)) also yields an impossible figure photograph. However, they all only view as an impossible figure object at a specific view (or projection). In other words, the objects produced cannot keep the impossible figure object structurally consistent in all views.

2D Spatial Tile from Impossible Figures. Tile-based analysis on 2D impossible figures are well conducted. Diego Uribe [86] discovered that an impossible figure with structures made of square bars blended at right angles can always be decomposed into a set of tiles which are referred to as impossible tiles. Figure 3.4 and 3.5 show two examples of the set of impossible tiles and their corresponding constructed impossible figures. However, one single tile still sheds no clues for us to reconstruct the 3D geometry that is consistent to the impossible figure structure.

From Diego Uribe [86], we can also understand that basic tile-based impossible figure constructed from two main elements: bars and joints, and rendered under isometric projection. Diego Uribe [86] proposed that we can have two different approaches when designing the set

Figure 3.5: (a) Another set of tiles proposed by Diego Uribe [86] for constructing impossible (possible) structures that contains two different views, so that parts of the figure are seen from above while other parts are seen from below; and (b) an example impossible figure constructed using the set of tiles shown in left.

Figure 3.6: (a) A joint (possible object) centred in an undefined plane-tiling polygon exists in six versions, one for each orientation; and (b) a catalogue of all available joints in which bars abut on a point. Taking account into orientation, there are 64 different joints in total.

of tiles, namely bar-oriented or joint-oriented, as shown in Figure 3.6 (a) and (b) respectively. Making use of the *local possible* concept from Wu et al. [96], all the bars shown in Figure 3.6 (a) are indeed possible objects that can be easily 3D reconstructed. However, works and studies on completely reproducing the entire 3D impossible object are very limited. The most extensive study prior to this work is Wu et al. [96] in which they produced a novel valid 2D view of an impossible figure by view-depending 3D modeling approach which keeps deforming a sur-

face mesh subjected to view changes. Such surface mesh is still a disconnected surface mesh like the paper model shown in Figure 3.3 (d).

Goal. Given an 2D tile-based impossible figure as an input, our goal is to define a tile-based geometry representation of the impossible world projected on the input impossible figure, which can fully contain the local structure whilst maintaining the global structural inconsistency. This is a challenging feat for two main reasons. First, the geometry of any local part of the resulting 3D models should conform to the corresponding local (possible) structure in the given impossible figure. Second, without twisting and disconnection (like Figure 3.3 (b&d)), we have to find a way to model an object that cannot be produced in the physical world, i.e. no real objects can serve as our reference.

To deal with this problem, we present a novel approach that can reconstruct a set of 3D model that is a *3D structural representation of impossible figures*. Specifically, the presented method analyzes line-art images of impossible figures and reconstruct them into box-based 3-D structures. Such a result cannot be achieved by any existing computer graphics methods for modeling tile-based impossible figures. Note that We do not consider any other classes of impossible figure that are not tile-based in this work. For example, the *disappearing space* class does not have a clear boundary between the foreground and background, so it is infeasible to reconstruct a valid 3D representation of the figure. Thus, we have an assumption indicted from the tile-based impossible figure properties: the input impossible figure is an orthogonal projection with three major axis directions in image space.

3.2 Related Works

Extracting the geometry information from an impossible figure and representing it in 3D space can indeed be considered as a special case of reconstructing 3D geometry from a single image.

In this section, we will review and discuss related works on 3D reconstruction techniques from a single image.

Photometry-based method. Horn [32] first introduced shape from shading, a technique in computer vision that reconstructs a 3D object (surface) from a single RGB image by estimating its surface normal under the assumption that the object being reconstructed is Lambertian object with single albedo shaded under distant lighting. Zhang et al. [101] and Durou et al. [19] best surveyed various shape from shading technique in the past two decades. Compared with shape from shading, our approach specialized in impossible figure without proper shading. Since our work nature is using a different approach, we refer the interested reader to the above two surveys for a better understanding on shape from shading.

Geometry-based method. Reconstructing the 3D geometry from a single RGB image is challenging without the depth estimation from pixel correspondences provided by multiple images. Oswald et al. [67] best summarized geometry recovery technique in the past decade specialized for curved objects from a single RGB image. Shtof et al. [79] developed an interactive system for modeling 3D objects from sketches figure. Inspired by the work of Shtof et al. [79], Chen et al. [8] developed another iterative system to extract 3D object from a single RGB image. All the above techniques fail in extracting structural consistent 3D objects from impossible figures due to the confusing depth misalignment nature of the impossible figures.

Figure 3.7: Our algorithm (a) takes an impossible figure (RGB image and normal map) as input, (b) converts it into a set of triangles or primitives where the geometric relationship are encoded by a relative position graph, and then (later in Chapter 4) (c) adapts it for different applications. Note that the face normal of the triangles in (b) is retrieved directly from the normal map, for illustration consistence, we use the same color coding from the normal map for the normal arrows. Please refer to Section 3.3.1 for a detail explanation on the relative position graph.

Figure 3.8: (a & b): our simple rotating-cube program for finding normal map colors of impossible figure images. (c) we then use GIMP to fill corresponding regions in the figure image.

3.3 Constructing Box-based Solid 3D Tiles

3.3.1 Inputs and Pre-processing

Our approach takes an image-based representation of an impossible figure as input, which consists of an RGB image and a corresponding normal map (per-pixel normal vector), see Figure 3.7(a). Such impossible-figure image can be easily obtained from the Internet, while its normal map can be easily prepared with the help of existing photo editing tools. In essence, to prepare a normal map image, we developed a simple program that renders a cube in orthogonal view with faces colored based on their normal vectors in the screen space (see Figure 3.8(a&b)). By rotating the cube until its orientation matches that of the given impossible figure, we can find the normal map colors for the impossible figure, and then use GIMP to flood-fill each surface region of the impossible figure accordingly (see Figure 3.8(c)).

i) Geometry Representations. Given the impossible-figure input, our next step is to convert it into a geometry representation, see Figure 3.7(b). Since different kinds of geometry representation could be needed for different working scenarios, we support two kinds of geometry representations:

- *Image-based triangles* (Figure 3.7(b) (top)): Here we construct a gradient-aware tri-

Figure 3.9: Partition the foreground region of the input impossible figure into quadrangles: (a) an input impossible figure, and the three major axes in orthogonal projection; (b) extract contour lines and identify their directions; (c-e) extend the contour lines one by one in the image space to partition the figure, see (e) for the result and Algorithm 1 for the pseudo code outline.

angulation (based on the gradient of the RGB image) on the image space of the input impossible figure, where each triangle is locally planar and consistent with the normal map. To avoid long and thin triangles, this mesh is further subdivided and re-triangulated to produce a finer mesh, see again Figure 3.7(b).

- *Solid Primitives* (Figure 3.7(b) (bottom)): Other than image-based triangles, which are surface-based open meshes, we developed an automatic method (see Section 3.3.2) to convert the input impossible figure into solid 3D primitives. The output here is a set of connected boxes, each with a position (relative to each neighbor) and an 3D orientation in the image space, see again Figure 3.7(b).

Note that both geometry representations should not contain relatively large components, i.e., size comparable to the whole model. This is required since the dynamic modeling method needs to progressively reconnect components to avoid gaps in user views, see Section 4.3. Hence, subdividing large components in this step allows greater flexibility in the reconnection.

ii) Relative Position Graph. Next, to facilitate real-time processing of impossible-figure structures for supporting various working scenarios, we created the relative position graph:

- *Nodes* (denoted by $\{P_i\}$) in the graph represent individual triangles (for the case of

image-based triangles) or primitives, e.g., boxes (for solid primitives); and

- *Edges* connect neighboring triangles/primitives based on the impossible-figure structure. For each edge (P_i, P_j) , we pre-compute the following information: For the case of solid primitives, we compute the vectors between the centroids of P_i and P_j : $\vec{d}(P_i, P_j)$ from P_i to P_j in the virtual 3D space. For the case of image-based triangles, we compute the relative distance between the centroids: $d(P_i, P_j)$, in the 2D image space. Additionally, we also pre-compute the angle between the normal vectors of P_i and P_j , i.e., $a(P_i, P_j)$; note that the normal vectors are retrieved from the input normal map. Figure 3.7(b) shows a zoom-in illustration of the nodes and edges relationship of a pair of example P_i and P_j .

3.3.2 Constructing Box-based Solid 3D Primitives

We developed an automatic method to arrange and tile connected 3D boxes over an impossible-figure image, see the result in Figure 3.7(b)(bottom). Recapping the assumption that the input impossible figure is an orthogonal projection with three major axis directions in image space, see the RGB arrows in Figure 3.9(a). Hence, we can fit axis-aligned boxes to match the local geometry of the image. There are two major stages in our method:

Stage 1: Partition the Impossible Figure

First, we partition the foreground of the input impossible figure into quadrangles in the image space, see Algorithm 1:

Lines 1-3: Extract Contour Lines. Here we locate the foreground image region, examine the gradient in the normal map, and identify contour lines in the impossible figure (see Fig-

Figure 3.10: (a) Occluded linkage: the dotted line between the two visible blue lines; (b) Star-junction (orange dots), where six contour lines meet and a cube is hidden behind.

ure 3.9(b)). By further analyzing the line orientations, we can find the three major axis directions in the orthogonal projection (Figure 3.9(a)) and assign a direction to each contour line.

Lines 4-29: Extend Contour Lines. Secondly, we partition the impossible figure by extending the contour lines one by one over the image space. This is done by extending the contour lines from their endpoints until the endpoints meet at a Y-junction (see Figure 3.9(b)) or reach the foreground boundary:

- (i) *Lines 7-15: Occluded (Layered) Linkage.* For geometric structures that are occluded by others in the image space, we have to recover them, so that the relative position graph to be built later can be a connected structure that represents the impossible figure.

One such case happens at a T-junction, see Figure 3.10(a): when we extend a contour line from endpoint p towards to the T-junction, we have to look for a parallel line matched with another T-junction. If found, there is hidden geometry in-between, and we should join the two lines and update p to be the far endpoint of the other contour line. By this layered representation, we can reconstruct the hidden geometry later in stage 2.

- (ii) *Lines 16-21: Join parallel lines.* When extending a contour line, if its endpoint meets

Algorithm 1 PARTITIONING THE IMPOSSIBLE FIGURE

```
1: Identify all contour lines in the impossible-figure image
2: Identify three major axis directions among contour lines
3: Assign a direction to each contour line
4: for each contour line do
5:   for each endpoint  $p$  of the line do
6:     while  $p$  not extend out of the figure foreground and not at a Y-junction /* See Figure 3.9(b) */ do
7:       /* Case 1: Occluded Linkage (See Figure 3.10(a)) */
8:       if  $p$  is at a T-junction then
9:         if a contour line is found in  $p$ 's extend direction then
10:          join the two lines
11:          update  $p$  as far endpoint of line found
12:        else
13:          break /* end while loop */
14:        end if
15:      end if
16:      /* Case 2: Join lines (See Figure 3.9(b&c)) */
17:      if  $p$  touches a parallel contour line then
18:        join the two lines
19:        update  $p$  as far endpoint of line found
20:        continue /* goto step 6 */
21:      end if
22:      /* Case 3: Change direction (See Figure 3.9(d)) */
23:      if  $p$  is about to extend to a perpendicular face then
24:        modify  $p$ 's extend direction
25:      end if
26:      Extend  $p$  until it intersects/touches a contour line
27:    end while
28:  end for
29: end for
```

another contour line, we should join the two lines and continue the line extension from the other end of the joined line, see Figure 3.9(b&c), so we can produce a larger quadrangle.

- (iii) *Lines 22-25: Change line direction.* When extending a contour line, if endpoint p is about to move into a face perpendicular to the contour line, the line partition does not

Figure 3.11: (a) A single box. (b) orange circles mark partially-occluded faces (partial quadrangles) of boxes. (c) box-fitting results: we assign the same color to the faces of the same box.

make sense if we continue with the same line direction, see Figure 3.9(d). Rather, we should adjust the direction along the line which is extended to make it stay on the next plane it enters.

Lastly, we extract the (planar) quadrangles in the foreground image space of the impossible figure, see Figure 3.9(e).

Stage 2: Fit and Connect Boxes

Next, we fit and construct connected boxes in the image space and a relative position graph to represent the input impossible figure. Before revealing our method, we first present the following two observations:

- First, there are two kinds of quadrangles: *full quadrangles*, whose opposite edges are in parallel, see Figure 3.11(a), and *partial quadrangles*, see the marked quadrangles in Figure 3.11(b). Note that full quadrangles are simply fully visible faces of boxes in the impossible-figure structure.

<p>Pattern with 3 full faces</p>
<p>Pattern with 2 full faces</p>
<p>Patterns with 1 full face and 1 connected edge</p>

Figure 3.12: Three different groups of patterns for box-fitting; note that these patterns can be rotated during the matching.

- Second, there are four kinds of boxes: boxes with three, two, one, and zero “fully visible” faces (full quadrangles), see Figure 3.12 (top row), (middle row), (bottom row), and the star junctions in Figure 3.10(b), respectively.

i) Initialization. The box-fitting stage starts by searching for a Y-junction shared by three full quadrangles, see Figure 3.12 (top). If found, we create a *cornerstone box* with such a quadrangle. Otherwise, we look for a 2-face pattern, see Figure 3.12 (middle row) and fit a cornerstone box with the two full quadrangles.

ii) Box fitting. Then, for each invisible face of the cornerstone box, we attempt to fit another box from the invisible face (as a shared face), so that we can find a neighboring box (node) of the cornerstone box. Here we apply the 1-face and 2-face box patterns shown in Figure 3.12 to reconstruct a neighboring box. Note that in the relative position graph structure, each box is a node and neighboring nodes are connected by an edge.

iii) Breadth-first traversal. We iterate this box-fitting process with a breadth-first traversal until all visible quadrangles have been visited. However, such a traversal may stop at the star junctions, see Figure 3.10(b), since the hidden box behind a star junction cannot be recovered by the patterns shown in Figure 3.12. Hence, whenever we reach a star junction, we reconstruct the hidden box by using the three shorter edges around it.

iv) Subdivision. Lastly, after the traversal is complete, we avoid long boxes by subdividing them into smaller boxes, see Figure 3.11(c) for the result. This is to give greater flexibility to the dynamic modeling method to be presented in Section 4.3.

3.3.3 Limitations

Hidden geometry. When modeling impossible figures from images, our current method considers two common cases of hidden geometry: i) in-between known structures along a straight

Figure 3.13: (a) An input impossible figure with a covered region behind the wall (see arrow); there could be many possible ways of reconstruction; (b) extra user input is needed, if we want to reconstruct the hidden structure marked by the arrow; otherwise, our current method will leave it unconnected.

line (Algorithm 1: *lines 7-15*), and ii) hidden corner boxes behind star junctions (Stage 2: step ii - box fitting). It cannot handle arbitrary hidden geometry, e.g., see Figure 3.13(a): in this case, our current method cannot automatically reconstruct and connect the hidden structure marked by the arrow in Figure 3.13(b). In this situation, extra user input is needed.

3.4 Summary

This Chapter presents an image based method to i) reconstruct the 3D geometries of each local portion of an impossible figure, and ii) link the reconstructed geometries by a relative position graph which presents the structural information of an impossible figure.

Chapter 4

3D Navigation on Impossible Figures via Dynamically Reconfigurable Modeling

4.1 Introduction

The impossible figures designed by M. C. Escher [56] (see Figure 3.1(a) for the INFINITE STAIRCASE) have intrigued generations of scholars, artists, game developers, and entertainers. One popular example was the box office hit *Inception* [64] in 2010, where a set of staircases were carefully arranged (see Figure 3.1(b)) for a specific camera view, so that the staircases form the INFINITE STAIRCASE from the audiences' perspective. When Arthur (Joseph Gordon-Levitt) and Ariadne (Ellen Page) were walking on the INFINITE STAIRCASE, they kept seeing the same busy secretary collecting the fallen papers.

Such visual effect violates our normal physical perception and requires tedious manual preparation in camera planning and scene arrangement, e.g., see the two realistically-looking impossible figures in Figure 3.3(a&c); they were created and photographed at some specific views. Hence, for the scene shown in *Inception*, if we change the camera view, a gap would appear between certain steps, i.e., *breaking* the INFINITE STAIRCASE and the *impossibility perception* (see Figure 3.1(b)).

Impossible figures have been used as mazes in games, e.g., *Echochrome* [28]. However, all existing games, including the popular action RPG game *Diablo II* [6] and the recent award-winning game *Monument Valley* [87] (see Figure 3.1(c)) that we are aware of assume an axonometrically-projected view. So far, no robust algorithms have been formally reported for building a virtual 3D maze of impossible figures for free-style first person navigation. Such problem is challenging but essential for immersive 3D gaming with impossible figures.

In computer graphics, there have been several pieces of research dedicated to model and render impossible figures, e.g., allowing us to rotate an impossible figure and to see it from another angle. Owada and Fujiki [68] formulated a constraint solver using multiple meshes to obtain a rotatable 2D impossible figure, while Wu et al. [96] employed thin plate spline to model an impossible figure via constrained deformation. However, none of them considered 3-D navigation over a virtual world built from an impossible figure.

Our Goal. Here, we aim to develop novel geometric modeling methods for immersive 3D gaming with mazes built from impossible figures. By this, users (or game players) can move over an impossible figure with first/third person views as if it is a normal 3D world, and experience the figure’s *subtle* impossibility, similar to INFINITE STAIRCASE in *Inception*.

The word “subtle” refers to the perception of impossibility when one experiences (e.g., sees) an impossible figure. This is a perception of realizing the figure’s structural inconsistencies, which is not obvious with a quick glance on local parts in the figure, but could soon appear after we recognize certain structural inconsistency in the figure. Considering the INFINITE STAIRCASE in *Inception* as an example, one initially may not notice the structure’s subtle-ness by seeing some of its local parts, but could later find the staircase going either up or down forever after recognizing the overall staircase structure.

Our goal is challenging. First, we cannot employ existing computer graphics methods, e.g., Owada and Fujiki [68] or Wu et al. [96], since we cannot twist the geometry of an impossible

figure nor break it at fixed gaps, see Figure 3.3. Second, the virtual maze needs to inherit the structure of the impossible figure, so that when the user navigates in the maze, he/she may experience the *subtle* impossibility perception. Lastly, more than static objects, we need to consider gaming elements, e.g., how users interact with one another and with dynamic moving objects, over the maze. These issues were not studied in any previous work.

To address the above issues, we develop a novel approach to model and render impossible figures. More specifically, it has the following contributions:

- (i) We present a set of *guiding principles* for producing seamless 3D navigation and interaction experience when building a gaming maze from an impossible figure (Section 4.2).
- (ii) We develop *dynamically-reconfigurable modeling*, which takes box-based 3D geometry representation of the 2D impossible figure (see Chapter 3) as input, and dynamically re-connects and re-structures an impossible-figure maze based on the game player’s view, so that we can achieve the guiding principles and enable seamless single-player navigation and interaction (Section 4.3).
- (iii) Lastly, we adopt and extend the basic modeling technique for different game maze types (Section 4.4): i) *open space*, ii) *a maze of corridors*, and iii) *multiple platforms*. To show the feasibility of our methods, we develop a game prototype for each of them.

4.1.1 Related Works

Before going into details on the novel approach, we first review and discuss computer games that employs impossible figures into their game levels.

Computer Games with Impossible Figures. Alexeev’s impossible world website [3] best summarizes computer games that feature impossible figures. Among these games, we discuss some of those that employ impossible figures for gaming. A Flash game called *Adynatopia* [53]

allows a player to move on a 2D static image of an impossible figure, and to play with its confused depth perception. The popular game *Diablo II* [6] embedded an impossible figure as a part of a huge 2.5-D gaming terrain in a level called *Secret Sanctuary*. Though both games used impossible figures in some interesting ways, the viewing angle of the axonometric projected camera remained unchanged when displaying impossible figures; and unlike most conventional 3D games, 3D camera controls are not available. A recent game on mobile platforms called *Monument Valley* [87] (see Figure 3.1(c)) allows us to rotate and play with a 3D object that was created by breaking an impossible figure at fixed gaps. This is similar to the paper model shown in Figure 3.3(c&d). Though it plays with impossible figures in a novel way, we can only see an impossible figure at a preconditioned view rather than walking and navigating over the figure. In contrast, this work, for the first time, constructs and provides a 3D gaming maze from an impossible figure. Using our approach, the player can walk on an impossible-figure maze with first/third-person 3D perspective viewing; and also control the camera during the immersive navigation.

3D Dynamic and Procedural Modeling. Parish and Müller [69] proposed *CityEngine*, an L-system-based procedural modeling engine capable of generating a large-scale virtual city. After that, Wonka et al. [95] developed a grammar-based method to automatically generate complex buildings, while Müller et al. [63] improved it with a shape grammar, especially considering façade details. Later, Lipp et al. [50] proposed an interactive visual editing paradigm for shape grammars. More recently, Danihelka et al. [16] developed a real-time procedural modeling method that can support multiple users navigation with consistent views among the users. Steinberger et al. [84] proposed a GPU acceleration method for generating and rendering an infinite city, where visibility information is integrated into the grammar evaluation process. These works aim to procedurally generate conventional virtual environments, which can be dynamically varied given different inputs. In contrast, we present a novel method to dynamically reconstruct a 3D gaming maze from an impossible figure with immersive 3D navigation and

interaction. Our solution takes user location and viewing information as input for keeping the maze consistent among the views of all the users.

Figure 4.1: A key frame sequence (following the orange arrows) in 3D animation *Hallucii*, showing a drunk man who walks over a 3D infinite staircase. The first image in the sequence shows a local view on the drunk man while the others show the entire staircase.

Figure 4.2: Another key frame sequence from *Hallucii*, showing that the drunk man finds the *same* briefcase again at the *same* corner after completing one cycle down the 3D infinite staircase. The yellow arrows indicate his walking direction.

4.2 Guiding Principles

Before presenting the guiding principles and our method for supporting 3D navigation over impossible-figure mazes, this section first revisits the notion of impossible figures and looks at a conventional 3D animation to explore how one would navigate in a 3D virtual world of an impossible figure.

4.2.1 Conventional 3D animations

Navigating (via *a game character*) through a 3D virtual world built from an impossible figure is analogous to looking (via *our eyes*) over a 2D image of an impossible figure. When navigating within a small local part in such a 3D virtual world, our 3D perception of the world works well since we can always reconstruct and interpret a normal (possible) 3D world locally. However, when we walk through *a larger 3D region* in such a virtual world, certain geometric contradiction, or subtle structural inconsistency, could arise when we attempt to interpret the full 3D geometry of the virtual world that we have experienced.

Taking *Hallucii* [90], a conventional 3D animation, as an example, see Figure 4.1. This animation features a drunk man walking down a 3D infinite staircase, which appears to be ordinary when the camera focuses on a small portion of the staircase (see Figure 4.1 (top-left)). During the walk, the drunk man kicks a glass bottle downstairs (see top-left image again), but later, he was dramatically hit by the same bottle from above, see the orange arrows in the figure for the animation sequence. Furthermore, he also finds the *same* briefcase again at the *same* corner after completing one cycle down the staircase (see Figure 4.2). This certainly violates our normal perception in the physical world, but demonstrates how one would experience in a 3D virtual world built from an impossible figure.

4.2.2 Guiding principles

From the ways one would see and explore virtual worlds built from impossible figures, e.g., *Hallucii* and *Inception*, we summarize a set of guiding principles to describe the perceptual experience that one would have in the exploration. These principles help us with the development of our computational method.

- *Maze Geometry*: First, the geometry of any local part of the maze should conform to the corresponding local (possible) structure in the given impossible figure, see Figure 4.1 (view 1) for a local view and Figure 4.1 (view 2) for a view of the entire impossible figure.
- *Seamless Viewing*: Second, the 3D maze should appear to be a normal (possible) 3D object in both first and third person views during the virtual navigation, see Figure 4.1 (view 1) and Figure 4.2. In other words, we should not see any gap or convolved surface like Figure 3.3(b) and 3.3(d).
- *Object Bondage*: Third, since we cannot define a global 3D space for the entire maze, 3D elements on the maze, including objects and avatars, should be bounded to associated local space, i.e., local parts of the maze. Thus, when the avatar walks over the maze, e.g., we may find the same object indefinitely at the same location (e.g., the briefcase in *Hallucii* (Figure 4.2) and the secretary in *Inception*) even though the path that we walk over is structurally-impossible in normal 3D space.
- *Subtle Impossibility Perception*: Lastly, since the maze is built from an impossible figure, one may subtly experience certain structural inconsistency when walking over it (e.g., the drunk man in *Hallucii* seems to be descending but is actually cycling), as well as interacting with other avatars or objects in the virtual world (e.g., in *Hallucii*, the glass bottle, which was kicked downstairs by the drunk man, could later hit him from above).

Table 4.1: Notations and functions used in Algorithm 2.

Notations/Functions:	Meaning:
P_c	the specific primitive containing the point of interaction, e.g., the current player location
$d(P_i)$	traversed distance along the path from P_c to P_i over the relative position graph
$p(P_i)$	parent of P_i during the traversal
$\mathbf{Nb}(P_i)$	set of neighboring nodes of P_i in the graph
$\text{SORTQUEUE}(Q)$	a procedure to sort the primitives in Q in ascending order of $d(P_i)$
$\text{SNAPVERTICES}(v)$	a procedure to snap together all vertices with the same image coordinates as v if their world coordinates are near one another; this helps avoid tiny gaps in the 3D construction
For the case of image-based triangles:	
$\hat{n}(P_i)$	normal vector of P_i
$a(P_i, P_j)$	given angle between $\hat{n}(P_i)$ and $\hat{n}(P_j)$
$\vec{e}(P_i, P_j)$	vector along the edge shared by P_i and P_j in the image plane
$\text{CONNECTTRIANGLE} \quad (P_i, P_j)$	a procedure to position and connect P_i to P_j in 3D: i) compute $\hat{n}(P_i)$ by rotating $\hat{n}(P_j)$ about $\vec{e}(P_i, P_j)$ by angle $a(P_i, P_j)$; ii) find P_j 's shared vertices with P_i and assign z of P_j 's vertices to z of the related vertices of P_i ; and iii) for P_i 's vertices (say v) without z assignment, we determine v 's z value by $\hat{n}(P_i)$ (from step i) and the 3D positions of the other vertices (from step ii). Note: X,Y denote the image space while Z is the out-of-the-paper direction (see Figure 3.7).
For the case of solid primitives:	
$\vec{c}(P_i)$	P_i 's centroid in 3D space relative to P_c

4.3 Dynamically Reconfigurable Modeling

Using the geometry representation from Chapter 3, we can dynamically reconfigure and construct a maze geometry of the impossible figure, which is a *view-dependent geometry*, subject to user's *viewpoint*.

Algorithm 2 presents the pseudo code of our modeling method given P_c , see also Table 4.1

Algorithm 2 GENERATING THE DYNAMICALLY-RECONFIGURABLE MAZE SUBJECT TO P_c

```

1: Queue  $Q = \{ P_c \}$ 
2:  $d(P_c) = 0$  and  $d(P_i) = \infty$  for all  $P_i \neq P_c$ 
3:  $p(P_c) = \text{undefined}$  /*  $P_c$  is root */
4: position  $P_c$  as it is in 3D
5: while  $Q$  is not empty do
6:   /* step 1: connect  $P_f$  to parent of  $P_f$  (if any) */
7:    $P_f = \text{pop front of } Q$ 
8:   mark  $P_f$  visited
9:   if  $p(P_f)$  is defined then
10:    if the case of image-based triangles then
11:      CONNECTTRIANGLE(  $P_f$  ,  $p(P_f)$  )
12:    else
13:       $\vec{c}(P_f) = \vec{c}(p(P_f)) + \vec{d}(p(P_f), P_f)$ 
14:    end if
15:  end if
16:  for each vertices of  $P_f$ ,  $v$  do
17:    /* vertex amendment: avoid tiny gaps */
18:    SNAPVERTICES(  $v$  )
19:  end for
20:  /* step 2: add  $P_f$ 's unvisited neighbors to  $Q$  */
21:  for each  $P_i \in \text{Nb}(P_f)$  do
22:    if  $P_i$  not visited then
23:      if the case of image-based triangles then
24:         $Distant = d(P_f) + d(P_f, P_i)$ 
25:      else
26:         $Distant = d(P_f) + || \vec{d}(P_f, P_i) ||$ 
27:      end if
28:      if  $d(P_i)$  is greater than  $Distant$  then
29:         $d(P_i) = Distant$ 
30:         $p(P_i) = P_f$ 
31:        if  $P_i$  does not exist in  $Q$ , push  $P_i$  to  $Q$ 
32:      end if
33:    end if
34:  end for
35:  /* step 3: sort primitives in  $Q$  */
36:  SORTQUEUE(  $Q$  )
37: end while

```

for the meaning of the notations in the procedure. Note that P_c is a specific primitive (in the relative position graph) that contains the point of interaction, e.g., the component on which the game player's avatar is located (see Sections 4.4.2.1, 4.4.2.2 and 4.4.2.3). To achieve the guiding principles (in particular, *maze geometry*) for these working scenarios, we develop the following key idea in constructing the dynamically-reconfigurable maze:

Continuously Shifting Gaps over the maze. When we construct a rigid (not twisted) 3D solid for an impossible figure in the physical world, such as the one shown in Figure 3.3(c&d), we have to break certain location(s) in the impossible figure and introduce gap(s) in its physical construction. The same applies to the construction of a virtual 3D maze for an impossible figure: *there must be certain gaps* in the 3D maze model since we cannot twist the maze model (see Figure 3.3(a&b)) and have to maintain the maze's geometrical rigidity.

However, since we deal with interactive scenarios, the maze geometry does not need to be static with fixed gaps. Rather, we can *continuously shift the gaps over the impossible figure* (i.e., relative position graph) and *keep them far away from the avatar* when the avatar moves over the maze. In this way, the game avatar can never reach the gaps in the gaming maze, and yet can seamlessly walk over the entire impossible figure. This is one of the key ideas in this work.

To achieve this, the general framework of Algorithm 2 is to dynamically construct the 3D maze starting from the avatar location (i.e., P_c), progressively connecting nearby primitives (which could be image-based triangles or solid primitives) to P_c , and expanding the maze model (P_c 's dominion) until it includes (visited) all the nodes in the relative position graph. To keep the gaps far away from the avatar, we use $d(P_i)$ to keep track of the traversed distances to every node from P_c over the graph, and in each iteration of the main while loop, we pick the primitive with the smallest $d(P_i)$ (after sorting) and connect it to the maze model. Since a gap is in fact an unconnected edge during the relative position graph traversal, after completing Algorithm 2,

Figure 4.3: 2D visualization of the dynamically reconfigurable model generated from Algorithm 2 using the image-based triangles representation of the PENROSE TRIANGLE. The blue dots in (a-d) indicate the avatar location (P_c); the grayscale shading of the triangles indicate the screen-space z value, where the darker the shading, the smaller the z value. Note that the gaps are located at the discontinuity of the grayscale shading as shown in (a) as an example.

Figure 4.4: Endless runner game prototype: (a) the input impossible figure (top) and the maze model dynamically reconstructed by Algorithm 2 w.r.t. P_c (green dot); (b&c) the starting and ending views, respectively, where the game avatar keeps running upwards endlessly to collect the coins on the impossible-figure maze; (d) without the adaptation strategy, the gap can be seen when the game player looks around; and (e) the adaptation strategy can help avoid the gap and achieve *seamless viewing*.

these unconnected edges should be far away from P_c , thus keeping the gaps far from the avatar (see Figure 4.3 for 4 examples). Furthermore, when the avatar moves, we update P_c , and apply Algorithm 2 again with the new P_c to continuously shift the gaps over the relative position graph. Therefore, we can continue to keep the gaps far away from the avatar.

Numerical issues in Connections. In step 1 of Algorithm 2, when we progressively construct the maze model by connecting P_f , we have to avoid producing tiny cracks in the geometric connections among primitives due to numerical errors when transforming the vertices. Hence, we introduce the SNAPVERTICES procedure at the end of step 1 to merge together nearby vertices between P_f and its neighbors.

Performance. Algorithm 2 is designed to run at ultra-high speed because its time complexity is linear in the number of primitives/nodes in the relative position graph. We evaluate its performance on the PENROSE TRIANGLE (image-based version) with 792 2D triangles. Each update takes only ~ 0.00048 seconds (averaged over 100 runs at different P_c) on a computer with Intel(R) Core(TM) i7 CPU 960 3.20GHz and 9GB memory. With such a good performance, commodity laptops can have sufficient computing power to support the user interaction even for heavy real-time 3D gaming applications.

4.4 Adaptation to Different Scene Types

The last part of our approach adapts and extends the basic modeling procedure presented in Section 4.3 for seamless viewing, and for various generic maze (terrain) types: 1) a maze of corridors, 2) open space, and 3) multiple platforms.

Here, we use Algorithm 2 to dynamically generate the maze (broken) geometry subject to the avatar, e.g., the red box in Figure 4.5(b), so that we can fulfill the *maze geometry* principle and the avatar can seamlessly move over the entire maze. However, when the user looks around the virtual world, he/she may *see* the broken gap somewhere in the maze at certain view directions (see Figure 4.4(d)), thus violating the *seamless viewing* principle (see Section 4.2). Below, we will describe how we overcome this issue in general and then discuss our adaptation to various generic maze types. Note that all the results presented here are box-based models (see Figure 3.7(b)) except Figure 4.4 and Figure 4.7, which are surface-based models mimicking the mazes in Hallucii [90] and a specific scene in Diablo II, respectively.

4.4.1 Achieving Seamless Viewing

Our key idea to achieve *seamless viewing* is by modifying the broken geometry from Algorithm 2, i.e., by pushing the gaps out of user's view frustum. However, we still keep the broken geometry from Algorithm 2 for supporting the navigation, and the modified version of the maze is only for rendering.

Here is our procedure to modify the maze geometry. First, we identify the gap(s) and their related nodes (next to the gaps) in the relative position graph. Recapping on gap(s) identification (see Section 4.3 and Figure 4.3(a)), gap(s) occurs at the disconnecting edges during the relative position graph traversal. Then, we copy the geometry of the missing node next to each visible node at the gaps to cover the gap. Lastly, we look for further visible gaps neighboring to the new node in a breadth first traversal manner; if found, we apply the steps above to the next gap(s), etc. By these steps, we can put the gap(s) away from user's view, and achieve *seamless viewing* (Figure 4.4(e)) upon view changes.

4.4.2 Different Generic Scene Type

4.4.2.1 Type 1: A Maze of Corridors

Since we can take solid primitives as inputs, we can produce a maze of corridors, where the game players navigate internally inside the solid primitives as walls, floor, and ceiling.

Game Prototype: chasing game. To show how our approach works with this type of maze, we built a chasing game prototype using the impossible figure shown in Figure 4.5(a). By our method, the user can never reach and see gaps in the scene in the navigation (Figure 4.5(c-e)) even though the scene is actually broken somewhere (see Figure 4.5(b)). Moreover, to promote the *object bondage* and *subtle impossibility perception* principles, we put some landmark objects in the scene, e.g., the goddess sculpture at the starting location of the game. Hence, when

Figure 4.5: Chasing game prototype: (a) input impossible figure (box-based solid primitives); (b) the maze model built from Algorithm 2 w.r.t. P_c (red box); and (c-e) subsequent screenshots, demonstrating *subtle impossibility perception* when we move over the maze, see also the mini-map (lower-left).

the user moves over the scene, he/she may return to the same goddess sculpture even the path that he/she has travelled is *physically impossible in 3D*.

This game prototype shows that our method offers 3D navigation that is (locally) consistent to that in a normal 3D virtual world, without implementing specific teleport tricks and non-Euclidean connections as in Portal and Portal 2 [88], and Antichamber [2], respectively. In Portal, the user has to shoot at the walls in the scene to create teleporting holes for the avatar to move across the virtual world in a physically-impossible manner; though this idea is innovative,

Figure 4.6: (a,b) our box fitting method can also work with impossible figures of the depth interposition class. (c) by manually connecting the interposition links with invisible boxes (see the double-ended arrows in (a,b)), our method can also work with this class of impossible figure.

there is no dynamic scene reconstruction and the scene is essentially a possible object attached with pairs of teleport holes.

4.4.2.2 Type 2: Open Space

Another maze type is open space, where the virtual pathways that the avatar walks through have no walls and ceilings. Careful reader may notice that in this open space environment, if the field of view of the game player is too large, it may be possible for the player to see a large portion of the virtual world, so we may not be able to avoid the gaps any more.

Here, we define a concept called *impossible loop*, which is a loop in the relative position graph, such that the geometry along the loop cannot be reconstructed as a valid *possible* object in 3D space. In the situation of open space mazes, if user's view covers an entire impossible loop, we cannot hide/shift the gap away from user's view any more unless with some special visual effects, e.g., fog. Please refer to the limitation section for detail. Below, we present two game prototypes of this maze type:

Game Prototype i): Endless Runner Game. The first one is an endless runner game, featur-

ing fast movement through an impossible-figure world, see Figure 4.4 for the screenshots and the impossible figure we employed. By using Algorithm 2, we can dynamically reconfigure the maze, so that no matter how far the game avatar runs, it can never reach the end, i.e., the gap. By our method, the game player can look around and gaps can be avoided accordingly in game player’s view (by shifting gaps out of the view frustum), see Figure 4.4 (d and e). Please see our supplementary video for the navigation experience.

Since the infinite staircase is repetitive by nature, an alternative way to model it is by duplicating multiple instances of its structure and vertically connecting them into a mega staircase structure. This could address the first three guiding principles. However, when the game player looks around with a not-too-small field of view, the *subtle impossibility perception* could break because he/she may find many duplicated stair structures co-existing in his/her view. Therefore, this duplication trick may not always work for impossible figures. Rather than using a static 3D model, we address this problem by a dynamic modeling approach, which offers higher flexibility in modeling the impossible-figure structure, since the dynamic model can adapt and respond to user’s view interactively.

Note that our method can also support mazes built with the “depth interposition” class (see Figure 4.6). This can be achieved by invisibly connecting the pair of interposition links with one or more invisible boxes in the relative position graph (see the double-ended arrows on Figure 4.6(a,b)). Hence, our dynamic modeling method will force an unnatural positioning between the scene geometries around the interposition links.

Game Prototype ii): Role-Playing Game. The other open-space prototype we built is a role-playing game environment that mimics the 2.5-D Arcane Sanctuary level in *Diablo II* [6]. Here, we use a third person view and the user can freely look around in the 3D virtual world, see Figure 4.7 (a-c) for screenshots with corresponding mini-maps, which indicate the avatar’s corresponding location and viewing direction.

Figure 4.7: Screenshots from the game prototype (Role-Playing Game) that mimics the Arcane Sanctuary level in Diablo II (which is simply a 2.5-D impossible world). *Zoom-in*: an example position and view direction of the avatar. (a-c): our method can reconstruct this impossible-figure world in 3D.

In this prototype, we purposely model the impossible figure structure, such that looking from certain locations in the scene, the user may see an entire impossible loop, e.g., see the mini-map in Figure 4.7(a), so the gap in the impossible loop cannot be shifted away from user's view. In this situation, we introduce a rendering effect with fog into the virtual world, so that the gap may be hidden by shifting beyond the fog region. Note that in this prototype, while maintaining the seamless viewing principle, we have carefully adjusted the size of the fog region, so that we can still maximize the amount of object parts in the impossible loop that can be seen by the user.

Figure 4.8: Multi-player shooting game prototype: The nontrivial impossible figure (a) from which the maze of corridors is built; and the maze model (d) reconfigured by Algorithm 2 w.r.t. P_c (the red box); Player 1 (in blue) took the lift to move up to a seemingly “upper” level (b). Next, Player 2 (in brown) entered the game and found Player 1 inside the lift above him (e). After adventuring through the impossible-figure maze for some time, the two players encountered each other in their own views, and then try to shoot at each other (c,f).

4.4.2.3 Type 3: Multiple Platform

We can also adapt our approach into multiple platform gaming environment; that is, to allow multiple users (or game players) to interact with one another simultaneously within the same impossible-figure maze. This problem is very challenging since the broken geometry produced from Algorithm 2 cannot be used to achieve the *maze geometry* and the *seamless viewing* principles for all the players in the same maze. However, we need a common 3D space in-between the players in order to accommodate the interaction among them.

Here, we demonstrate this challenge by the first-person shooting game prototype shown in Figure 4.8. For example, when a user shoots at the other, we have to ensure proper 3D calculation over the maze, even though the maze is modelled in the form of an impossible figure. We propose certain adaptation strategies to allow multiple players to simultaneously move and interact with one another, apparently in the same 3D virtual world built from a nontrivial impossible

figure, e.g., see the top-left of Figure 4.8.

Achieving Multi-player Interaction. We propose an egocentric approach to resolve the above issue: i) we first apply Algorithm 2 to create a maze geometry locally for each game avatar; and ii) when considering Player 1, we duplicate an instance of Player 2 (and all others) in the local (broken) maze geometry of Player 1, so that Player 1 can see and interact with other players using his/her own maze geometry. Likewise, Player 2 interacts with Player 1 through duplicated instances of Player 1 in Player 2’s own (local) maze geometry.

Note that there is an assumption behind the second step. We assume that the virtual path along which Player 1 sees Player 2 (in Player 1’s own world) should be consistent with the virtual path along which Player 2 sees Player 1 in Player 2’s own world. Such a property can be achieved by our method because Algorithm 2 actually constructs a dynamic maze in the form of a *minimum spanning tree* over the relative position graph. Hence, the path along which Player 1 sees Player 2 is the shortest path from Player 1 to 2, so Player 2 can also see and interact with Player 1 along the same shortest path (but in reversed direction) in his/her own world.

Game Prototype: Multi-player Shooting Game. Figure 4.8 shows screenshots of our game prototype. We can see from the perspective of Players 1 and 2 in top and bottom rows of the figure, respectively. From the screenshots (see the supplementary video), although Player 1 has secured the lift to move up to a seemingly upper level, Player 2 can still reach him through the *structural impossibility* in the scene by moving horizontally over the impossible-figure maze. By then, the avatars of the two players can meet each other, and start the shooting interaction over the maze.

By our adaptation strategies for multi-player interaction, each user can move seamlessly in his/her own impossible-figure maze without reaching and seeing any gap. And at the same

time, they can see and interact with one another without breaking any guiding principle, even though they do not share the same maze model in the interaction.

4.5 Adaptation to Other Application

Although we focus our modeling method on gaming applications, we may also apply it to other applications.

4.5.1 Haptics Interaction Scenario

We may also adopt our method to construct a dynamically-varying 3D model of an impossible figure, so that one may touch and feel the surface of an impossible figure with haptic.

Adaptation Strategy. Our key idea here is to couple Wu et al. [96] with Algorithm 2. That is, we use Wu et al. [96] to produce plausible views of an impossible figure for users to rotate the figure, and use Algorithm 2 to enable users to touch and slide over the figure surface, as if it is a normal 3D object.

In detail, we couple the two methods as follows: i) we employ Wu et al. [96] to generate 2D views (god views) of an impossible figure upon view rotation; ii) we take the image-based representation (mesh) of the rotated figure from Wu et al. [96] and deform the mesh along the line of sight to make the mesh surface consistent with the rotated normal map; and iii) we apply Algorithm 2 to this mesh subject to the haptic stylus location as P_c . Hence, when the haptic stylus moves over the impossible-figure surface, Algorithm 2 helps to construct a dynamic geometry and shift the gap(s) away from the haptic stylus. By this, we can seamlessly slide over the impossible-figure surface, and achieve the *maze geometry* principle. Note also that the mesh produced from Wu et al. [96] cannot be used to support the haptic interaction since such mesh is highly twisted and convoluted in the 3D space.

Figure 4.9: An exemplary interaction scenario of using the haptic device to touch and feel an impossible figure PENROSE, whose structure was used to build the infinite staircase in *Inception*. Note that the 3D model (shown in the bottom) is broken most of the time (but not sensible by the users), as indicated by the abrupt depth discontinuities.

4.5.1.1 View-dependent modeling

The purpose of view-dependent modeling is to generate 2D views (god views) of an impossible figure upon view changes. We deform mesh vertices, rather than pixel locations as done in Wu et al. [96], to improve the performance. While the deformation strategy in Wu et al. [96] is adequate for viewing purpose, the resultant geometry can be so convoluted in the third dimension (though not observable on the screen), which violates the *terrain geometry* principle and makes it impossible to create the illusion of gliding over a rigid polyhedron surface. As such, we deploy the 2D result from Wu et al. [96], that is, the image coordinates of the deformed vertices, but reconstruct the 3D geometry on-the-fly (see below) to produce a mesh that is conducive to our interaction requirements.

4.5.1.2 Interaction Scenario

Let us first start with an example interaction scenario on PENROSE using one touch point haptic device like PHANTOM OMNI, see Figure 4.9. The corresponding 3D geometries, which are dynamically updated depends on the touching location (using Algorithm 2 regardless of viability of triangles) during the interaction, are also shown. Though the geometry is broken

all the time, as indicated by the abrupt depth discontinuities (the brighter the gray level, the smaller the depth value), the user can still see and touch on a ‘plausible’ polyhedral object, which can even be illuminated akin to possible polyhedrons.

- **Touch:** First, a user can hold the haptic stylus and lift it up to make contact with the 3D surface of an impossible figure;
- **Slide:** Upon contact, the user can slide the stylus of the haptic device to tactically feel the surface of the impossible figure; moreover, one can move the stylus over the entire figure;
- **Grab:** Using the switch on the stylus, the user can also grab the impossible figure with the stylus pen like picking up food with a fork; and while holding the switch, he/she can rotate the impossible figure by revolving the stylus;
- **Rotate and Slide again:** Upon releasing the switch, one can continue moving the stylus pen on the impossible figure even though it has been rotated;
- **Illuminate:** Lastly, one can also trigger the lighting mode by pressing and holding the stylus switch when the stylus pen is not touching the object surface. Thus, rotating the stylus can interactively illuminate the impossible figure at different directions.

Though the interaction scenario above presents our idea using a non-gaming based one point haptic device, such scenario can be easily extended for a gaming scenario, where the haptic stylus contact is replaced by an avatar character contact virtually and transfer the tactical feeling to the real world player by a haptic game controller. We are then going to discuss how this interaction scenario can provide a valid sensation.

4.5.1.3 Applying guiding principles for a valid sensation

The interaction scenario above echoes our *Maze Geometry*, *Seamless Viewing* and *Subtle Impossibility Perception* guiding principles, which can also be interpreted to produce convincing touch sensation on impossible figures by the following ways:

- when we glance over an impossible figure, the corresponding 3D structure at any sufficiently-small area around our eyes' focus can always be valid and polyhedral (*Maze Geometry*). Similarly, when the stylus moves on an impossible figure, we require of a seamless touch (*Seamless Viewing*), where the local geometry should behave like a physically-possible polyhedron.
- when we glance over a larger area on an impossible figure, careful readers may notice, along the eye-glancing path, certain (subtle) structural inconsistency, which is the figure's *Subtle Impossibility Perception*. This calls for an extension of such subtle perceptual experience from visional to touching.

4.5.1.4 Practical Considerations

Shifting in z : Since we randomly pick a triangle as P_c when the system starts, the range of z values in the reconstructed mesh may be unbalanced about the $z=0$ plane, thus limiting the visible range of movement if we keep our camera and clipping planes unchanged. Due to budget we employ only a mini haptic device with a relatively small movement space. Hence, upon applying Algorithm 2 for the first time, we shift the mesh along z and center it at $z=0$ for maximizing the visible interaction range. For applications which visual examination on the happening of z shifting is not applicable, one can consider translating the camera according to the current location of the impossible figure mesh.

Scaling in z : Driven by the same reason, we also scale and compress the z range of the reconstructed mesh. This is done by scaling up the z component of these raw normal vectors in

the normal map, and then normalizing them to unit vectors. Hence, the reconstructed mesh can be compressed like a bas relief to keep the sharp angles between planar surfaces (orientation discontinuity).

Multi-touch Contact Points: One may think of using hand based haptic device which supporting multiple finger based contact point, and though is very expensive for now, but highly probable to be available very near future in the game controller market with a relatively cheap cost. Again, owing to budget reason, we cannot do experience on such a device; however, given the nature fact that five fingers cannot be separated apart for a very long distance, we can always regard the group of fingers as one single contact point and apply our approach. Regretfully, there is no unique convincing solution when it comes to two hands sliding towards opposite directions. However, we argue that one hand interaction is already covering a bucket of manipulations that could enrich a game very much.

4.5.2 Virtual Reality Experience

For example, we may use it to construct a virtual world for animation production similar to that of *Hallucii*. Moreover, we may extend our method with stereoscopic viewing for supporting VR applications on Oculus Rift and Google Cardboard.

4.6 User Study

We conducted a user study to examine the feasibility of our approach. In detail, we would like to study the user experience when the participants navigate and explore the geometry reconstructed by our method.

Preparation. First, we slightly modified the multi-player shooting game prototype presented in Section 4.4.2.3 with the followings: i) instead of showing the whole maze, the mini-map on

Figure 4.10: A flattened figure resembling the impossible figure input in the multi-player shooting game prototype.

the gaming interface shows only a local portion of the maze surrounding the gaming avatar; and ii) we created a possible 3D maze (see Figure 4.10) that resembles the impossible-figure maze used in the multi-player shooting game prototype. This maze is created by flattening the impossible figure, i.e., replacing the vertical lifts with horizontal corridors.

Participants. We recruited 20 participants (volunteer-based) aged from 17 to 32: 13 male and 7 female. All of them have at least five years of experience in playing mobile or console games. We randomly paired them up into ten groups, each with two participants, and we performed the user study procedure described below in a group-based manner.

Procedure. When a group of participants came, we first gathered their personal information: gender, age, gaming experience (years), shooting game experience (years), 3D gaming experience (years), knowledge of impossible figures (poor 1 to good 7), experience in playing games related to impossible figures (list, if any), the ability of perceiving 3D objects from a 2D figure (poor 1 to good 7), and the ability of knowing direction in a 3D virtual world (e.g., will you

easily get lost in a 3D maze?) (poor 1 to good 7).

Tutorial session. Then, we present to the group the knowledge of impossible figures by using the contents in Section 4.2: “The Notion of Impossible Figure.” We then show the INFINITE STAIRCASE in Figure 4.1 (top-right) to the participants as follows: “An example would be the INFINITE STAIRCASE that is a two-dimensional depiction of a staircase in which the stairs make four 90-degree turns as they ascend or descend yet form a continuous loop, so that it gives an impossibility perception that one could climb them forever and never get any higher.”

Next, we show to them the impossible figure for building the maze in the multi-player shooting game, i.e., Figure 4.8 (top-left), and discuss with them to ensure that they understand the perceptual impossibility in the figure. The two participants in the group are then randomly assigned as Player A and Player B, sit on the opposite sides of a table, and try our multi-player shooting game prototype on his/her own computer screen.

Practice session. After the tutorial session, the two participants are given three minutes to individually try the game in a single player game. This allows them to get familiar with the gaming controls. In addition, we prevent them from understanding the maze structure by allowing them to explore only a very small local area around the starting location in the maze.

Gaming sessions. There are two gaming sessions for each group of participants: one with the impossible-figure maze in Figure 4.8 and the other with the possible 3D maze in Figure 4.10. The order of these two gaming sessions is randomly assigned but evenly balanced, so that five out of the ten groups tried the impossible-figure maze first and then the possible 3D maze, and the other five groups tried the other way around. Note that the participants do not know which maze they are trying.

Tasks. After the practice session, the first gaming session starts by randomly assigning a starting location to the avatar of each participant, where the two random locations are far away

from each other in the maze. Then, the two participants are given the following tasks:

- (i) Explore the virtual world.
- (ii) Find and shoot the other player (until no more hit point).

After one of the players finishes the tasks, we end the gaming session and proceed to a questionnaire session (see below) before the second session (same task but different maze).

Questionnaire. After each gaming session, we ask each participant to fill a questionnaire to examine their experience in the session. The questionnaire consists of three sections with variables designed based on Davis [17] using a Likert scale of 1 to 7 (very unlikely 1 – very likely 7):

a) Perceived Impossibility (1–7):

- a1) The maze I navigated is an impossible figure like the one shown to me.
- a2) I have always tried to mentally reconstruct a 3D picture of the virtual world I have navigated.
- a3) I think I moved to a higher (lower) ground, but then found I went back to the lower (higher) ground.
- a4) I am confused on which level I am currently located at.
- a5) I do not think I have been navigating on the same ground.

b) Perceived Ease of Use – Gaming Interface (1–7):

- b1) I would find the control of the game easy to get to where I want to reach.
- b2) I would find a mini-map only showing a local region helpful for me to understand the maze.

Figure 4.11: Statistical results gathered from the twenty participants who tried our gaming prototype: mean and standard deviation of variables (a1)-(a5) and (b1)-(b3) for the case of impossible-figure maze (left) and possible 3D maze (right).

b3) I would find a mini-map with a fixed viewing angle (i.e. not rotating, only translating) helpful for me to understand the maze.

c) *Other comments*, if any.

In the questionnaire, section (a) attempts to explore the navigation experience of the participants. Here, variable (a1) directly asks the participants on whether they find the maze to be an impossible figure or not; variable (a2) examines if the participants are aware of the mental process of reconstructing the virtual world, while variables (a3-a5) explores literally the experience of the participants when they navigate in the maze. We expect high scores (in Likert scale) in the five variables for the case of impossible-figure maze.

Section (b) aims to explore whether the gaming interface provides a fair platform for comparing the navigation with the impossible-figure maze and the possible 3D maze. The three variables in this section examine how individual user interface feature aids the maze exploration. Among them, variable (b1) focuses on the navigation control, while variables (b2) and (b3) focus on the mini-map. Here, we expect similar scores in the three variables for both mazes.

Results. Figure 4.11 presents the statistical results, showing the mean and standard deviation

of variables (a1)-(a5) and (b1)-(b3). First of all, from the plots of variable (a2) in the two cases (impossible-figure maze and possible 3D maze), we can see that the participants generally tried to reconstruct a 3D mental picture of the virtual world. Moreover, by comparing (a3)-(a5) across the two cases, we can see that the scores of these variables are generally higher in the case of impossible-figure maze, suggesting that the participants perceived higher degree of impossibility for the case of impossible-figure maze. Additionally, there are no major differences across (b1) to (b3) in the two cases, indicating that the participants are generally satisfied with the gaming interface and show no bias in the two gaming sessions between the two mazes.

Lastly, we performed a paired t-test on variable (a1) to examine whether there is a significant statistical difference on the perception of impossible-figure maze and possible 3D maze in the gaming sessions. The null hypothesis H_0 is *the mean values of variable (a1) in the two cases are equal*. After the computation, we found that the resulting t value is 6.565, which is larger than the critical value from the t-test table: 4.59 with degree of freedom $DOF=19$ and a significance level of 99.99%. Hence, we can reject H_0 at the significance level and suggest that there is a statistical difference between participants' perception on the two mazes.

4.7 Limitations

Input impossible figures. Our method takes input from the tile-based 3D reconstruction method described in Chapter 3, which inherits all limitations discussed in Section 3.3.3. Thus, we design our method exclusively for impossible figures related to depth: *Depth contradiction* (e.g., PENROSE TRIANGLE) and *Depth interposition* (e.g., IMPOSSIBLE CUBOID), see Figure 3.2. These are common classes of impossible figures used extensively in existing games.

For *Disappearing normals*, some of its image regions do not have consistent normals, e.g., see the plane marked with an arrow in Figure 3.2(b), this ambiguous plane could appear to be horizontal when seeing from top/bottom, or vertical when seeing from the right. Hence,

we cannot form a consistent geometry representation for working with this class of figures. For *Disappearing space*, due to its incomplete silhouette, there are no clear boundary for foreground and background, see Figure 3.2(c), so our method cannot form a consistent geometry representation for impossible figures of this class neither.

Avoiding gaps in user’s view. Our dynamic modeling method by geometry reconnection and gap shifting assumes that user’s view is always local to a portion of the impossible world such that the portion is a “possible” object. However, if the user is able to move to a certain location in the impossible world such that he/she can see a larger portion of the world that includes an impossible loop structure (e.g., a PENROSE TRIANGLE), then the gap cannot be hidden from his/her view, e.g., in the case of an open space maze without walls. To circumvent this issue, other than having walls, we may use some rendering effects such as fog at specific locations in the virtual world, or redesign the virtual world with a larger impossible-figure structure so that it cannot be seen within a perspective view.

4.8 Summary

This chapter presents novel computational methods that construct virtual mazes using impossible figures for 3D gaming. The method delivers seamless interaction and viewing with impossible figures, so that one can navigate through impossible-figure mazes in 3D gaming for the first time.

There are four contributions in this work. First, we revisit the notion of impossible figures, and propose a set of guiding principles for delivering seamless 3D navigation and interaction experience with 3D impossible-figure maze. Second, we devise a real-time procedure to dynamically reconfigure and reconnect the maze model (reconstructed from the work in Chapter 3) subject to the game player’s position, thereby achieving seamless navigation over

the maze even though a complete maze cannot be modeled plausibly in 3D (without gap and twist). Third, we adopt and extend the method to support various generic maze types, and develop several game prototypes to demonstrate the applicability of our method. Lastly, we adopt and extend the method to support other applications, including haptics and VR interaction.

Chapter 5

Tile-based Reconstruction of Surfaces

5.1 Introduction

Computer-aided-geometric design (CAGD) systems offer powerful solutions with accurate engineering constraints in the design and manufacturing of wide ranges of 3D geometric models. One emerging area is modern architecture, where the exteriors of a fast growing number of specially-designed building landmarks were using CAGD methods [71, 70]. These buildings typically come with noticeable aesthetic-looking curved surfaces, such as those in Figure 1.4. However, compared to the immense research effort that has been devoted to the design of professional CAD systems, until the time that this work is done, relatively little attention has been paid to geometric techniques to improve the quality and efficiency in architectural surface modeling.

Recalling from 1.1.3, modern architecture relies on prefabrication to assemble the exterior of a building or the façade. The spirit of prefabrication only utilized when the number of distinct prefabricated components (or the tiles) required to fully resemble the building façade is minimal. If we only apply basic symmetries and similarity to determine the set of tiles for building the façades for aesthetic freeform surfaces (e.g. those from [71, 70] or Figure 1.4), the number of tiles is usually large in order to maintain the complicated shapes of the aesthetic freeform

Figure 5.1: Overview of our approach over a MONKEY SADDLE surface: We iteratively cluster and analyze: For this example we (i) cluster the quads into 24 clusters, and (ii) analyze the edge connections to detect 33 distinct edge lengths in the specific clustering configuration. Then (iii) we optimize the given surface into a KT-surface.

surfaces; and thus, the construction cost is high and the construction efficiency introduced by prefabrication is minimal. The challenging problem here is how to determine the optimized set of tile in terms of number and shape. We introduce the K -set tilable surface to address this problem.

Given a quad-based surface as an input, the goal of our work is to generate a K -set tilable surface (or KT-surface) with a prescribed K while maintaining the KT-surface close to the input surface. Beyond being close to the surface, we expect the KT-surface to have similar geometric characteristics, such as the surface curvature, as in the input surface, and thus taking the square faces of a voxelization of surface is an invalid solution. Note, however, that we do not require the quads to be planar. The main contribution of this work is two-fold. First, the introduction of the notion of K -set tilable surfaces. Then, we present a solution to the problem based on non-linear optimization. We show the construction of various KT-surfaces that are quantitatively close to the input surfaces.

5.2 Overview

To generate a K -set of tiles, we take an optimization approach that consists of iterative clustering and analysis (see Figure 5.1): after clustering the quads, we generate K representatives

Figure 5.2: (a) original surface; (b) simply replacing surface quads with S-quads; (c) optimized S-quads tiled on KT-surface.

and learn the flexibility from the clustering by analyzing the relations between the clusters and the quad-mesh. The surface vertices are then optimized so that all quads agree with the representatives while remaining close to the given surface. We assume that the given surface is tiled with quads only, possibly initially of a unique shape. To reduce the number of distinct quad shapes to K , one can first cluster together the surface quads by their shape similarity, using some rotation-invariant similarity measures between quads, and replace each quad with a representative quad from the associated cluster. In the following, the representative quads are denoted by *S-quads*.

Simply taking the mean shape of quads in a cluster as the shape of an S-quad and replacing the surface quads by their respective S-quads will break the surface, because adjacent quads will no longer agree on their common edges, see Figure 5.2. Clearly, the geometry of S-quads cannot be determined merely from the clusters alone. The S-quad shapes must respect also the spatial constraints among their instances over the tiled surface to guarantee their proper connectivity.

In addition, one also has to ensure the shape consistency of the instances so that they have compatible lengths in diagonals and edges, as well as consistent signed volume occupied by the quad instances. Since we consider non-planar quads, the vertex arrangement of quads can affect the shape uniqueness in addition to the lengths. Lastly, one also needs to ensure that the surface tiled with the quads instances approximates the input surface.

Our optimization model solves for the vertex coordinates with various constraints that respect the edge connectivity and shape consistency among the S-quads instances, as well as the surface approximation. Consequently, we can iteratively optimize the input surfaces into K-set tilable surfaces.

5.3 Related Works

5.3.1 Research Works on Discrete Differential Geometry

Discrete differential geometry (DDG) [7] is an emerging research area where differential geometry [74, 18] interacts with discrete geometry. With the discrete equivalences of the geometric notions and methods in classical differential geometry, recently researchers have explored the potential of DDG for computer graphics modeling and applications. DDG allows formulating and optimizing various geometric properties in a discrete manner. Some recent examples are the construction of various models [57, 55, 78, 62, 73]. Another example where a model is analyzed and then optimized is the work of Gal et al. [27]. They introduce an editing mechanism that analyzes and edits man-made models by learning the inter-relation among the model parts. Our optimization tends to capitalize on the inherent symmetry of the given shape [97]. In that sense, a related work is the model symmetrization of Mitra et al. [59].

The optimization method that we present here is applied on quad-meshes. Quad-meshes draw more attention recently since they are attractive for modeling surfaces: the quad elements can

be nicely aligned with the principal directions of the surface. The work of Daniels et al. [15] simplifies quad-meshes. Our work also simplifies quad-meshes, but not in terms of the surface quad count.

5.3.2 Concurrent Similar Research Works

Concurrent to our work, there are two similar pieces of independent research work that share the same general motivation with us. All aim at optimizing the number of tile shapes required to construct a surface. The approach of Singh and Scott Schaefer [80] is similar to ours, but their tiles are triangles. They search for a clustering of triangles that can be optimized to approximate the given model. Starting from a single cluster, they keep adding clusters until the approximation is sufficient. The work of Eigensatz et al. [20] considers molds and panels rather than congruent tiles. Their focus is on the reusability of molds in fabricating panels for forming globally coherent surfaces.

5.4 Clustering

The purpose of the clustering phase is to group quads with similar shapes and define an initial set of representative S-quads. Then the shapes of these S-quads are optimized, so instances of them can then tile the KT-surface. First, we define the metric for computing shape dissimilarity between two quads p and q , whose vertices are p_i and q_i , respectively:

$$s(p, q) = \min_j \sum_i \|q_i - T_j(p_i)\| \text{ for } i \in [1, 4], \quad (\text{Eq. 5.1})$$

where T_j is a rigid body transformation involving only arbitrary translation and rotation. This definition searches for the best possible registration between the two quads among the eight different ways to correspond vertices from the two quads. Subsequently, we can construct an affinity matrix of quads, see Figure 5.3.

Figure 5.3: Affinity matrices (linearly color-mapped) for (a) MONKEY SADDLE and (b) SEASHELL surfaces.

5.4.1 Initial Clustering

The clusters are formed by a series of cluster merge, where at each step two clusters are merged until we are left with exactly K clusters. We start off with a large number of small clusters created simply by grouping together all quads with strong similarity defined by a prescribed threshold. The result of this step yields an initial set of N clusters. Once a set of clusters, or say a *clustering configuration*, is formed, the mean shape of each cluster defines an initial S-quad. The S-quad instances are now placed over the surface to replace the original quads. Since the quads have eight possible orientations, each S-quad instance is tagged with the orientation that maximizes its similarity with the quad tile it replaces.

To continue the discussion, the *weight* of a cluster C_i is defined as the total sum of all pairwise dissimilarity among its quads, and is denoted as $S(C_i)$. The cost of merging two clusters C_i and C_j is $S(C_i \cup C_j) - S(C_i) - S(C_j)$ and the weight of a clustering configuration is the sum of weight of the individual cluster inside.

However, the clustering configuration with the lowest weight is not necessarily the one that will incur the least constraints on the consequent KT-surface. The *quality* of a configuration is defined also by the degree of *flexibility* it imposes on the KT-surface. The “flexibility” is learnt

by analyzing the relations between the clusters and the quad-mesh, and the degree of freedom the S-quads definition imposes on the mesh edges. For that we next apply the edge connections analysis and determine its flexibility.

Analyzing the edge connections. Recall that the S-quads are templates of quads and their instances over the resultant KT-surface must have matched length along their shared edges. We thus examine the edge connections between S-quads over all edges on the input surface and formulate constraints for the optimization of S-quad shapes. To this end, constraints of individual edges are aggregated to form the *edge connection constraints* with the assistance of an *edge sharing graph*. Basically, it is an undirected graph, say $G = (V, E)$, where each vertex in V corresponds to one of the edges of the k S-quads, i.e., $|V| = 4k$. Two vertices in G are connected if their associated S-quad edges (for some S-quad instances) share an edge on the given surface.

Figure 5.4: Given four clusters of quads on MONKEY SADDLE (a), we analyze the edge sharing conditions for S-quad instances by an edge sharing graph (b). Edges marked with the same color should have matched edge length on the KT-surface (c).

Taking the 4-by-4 MONKEY SADDLE surface in Figure 5.4 as an example. The connections (edge sharing) between instances of S-quads on the surface define the connections between

related vertices in the edge sharing graph. Since the same S-quad may have more than one instance over the surface, instances of the same S-quad may sometimes be placed next to one another, and hence, some vertices in G may connect to vertices of the same S-quad. However, we can ignore edges that loop back to the same vertex in G since loops do not pose any constraint.

Based on the edge sharing graph, we define the edge connection constraint by searching over connected components in the edge sharing graph. Each disjoint subgraph, i.e., a connected component, in the graph is a group of S-quad edges that should have compatible edge length on the resultant KT-surface.

Following the example shown in Figure 5.4, the edge sharing graph consists of seven disjoint subgraphs, and hence there are seven distinct edge lengths in the KT-surface. In Figure 5.4, the graph vertices belonging to the same subgraph are labeled with the same color and these colors are applied to visualize the edge connection constraints over the surface. During the optimization, edges of the same color are constrained to have compatible length.

The larger the number of disjoint subgraphs, the higher the number of distinct edge lengths exist on the KT-surface. This number defines the degree of freedom in modeling a KT-surface. In other words, the number indicates the *flexibility* of a KT-surface. Now, the degree of flexibility can be directly determined by the clustering configuration; it serves as a useful indicator on the quality of the clustering, and is used in searching for the best configuration. In practice, we found that for some clustering configurations with a relatively large number of clusters, their degrees of flexibility may still be low due to poor arrangement of S-quads.

5.4.2 Cluster Merge

Recall that the clusters are formed by a series of merge operations. Each series of such binary merges yields a clustering configuration that is associated not only with cost, but also with a

degree of flexibility learnt from its edge sharing graph. Ideally, we would like to exhaustively test all possible series, and learn which clustering configuration yields the best KT-surface, but since our problem is intrinsically similar to the NP-hard optimization problem *Minimum K-clustering Sum* [4], this, however, is not computationally feasible and we employ the following heuristics: We sort all the initial N clusters by their weights $S(C_1) \leq S(C_2) \leq S(C_3) \leq \dots \leq S(C_N)$. Then, we subsequently pick two clusters from the front of the list and compute the merging cost as well as the change in degree of flexibility. Meanwhile, we keep track of 10 candidate configurations¹ and the lowest cost among them serves as an upper bound for the merging cost. Since the weight of the merged cluster is always larger than the sum of weights of the two individual clusters being merged, this upper bound is used for an early termination of an otherwise exhaustive search. In addition, we penalize the merging cost (with a user-specified parameter) if the merge results in severe reduction in the degree of flexibility. Notice also that upon each binary merge, we can amend the edge sharing graph rather than re-building it in order to learn the degree of flexibility. In the spirit of genetic algorithms for searching a very large space, the resultant ten candidate configurations are later used as the seed configurations to generate the next ten candidates when we further reduce K . The different series of binary merges yield candidate clustering configurations, or in fact, sets of S-quads.

5.5 Optimization

The KT-surface geometry is generated by optimizing the $\mathbf{v}_{i,j}$ vertices positions, under the constraints defined by the edge sharing graph. The vertices positions aim to remain closer to the original surface, expressed by the F_d term, which measures the distances between the corresponding vertices of KT-surface and input surface:

¹The number of candidate configurations is controllable as a parameter, and a larger number, such as 50, is needed for surfaces with more quads.

$$F_d := \sum_{i,j} \|\mathbf{v}_{i,j} - \mathbf{v}_{i,j}^0\|^2, \quad (\text{Eq. 5.2})$$

where $\mathbf{v}_{i,j}^0$ is the original position of vertex $\mathbf{v}_{i,j}$.

To preserve the surface smoothness, we employ the fairness term F_f and apply it to vertices that are not on sharp features, such as corners and crest lines. Based on [51], we use the following fairness term on interior vertices:

$$F_f := \sum_{i,j} \text{fair}(\mathbf{v}_{i,j}) \quad (\text{Eq. 5.3})$$

$$\text{fair}(\mathbf{v}_{i,j}) := \begin{cases} \|\mathbf{v}_{i,j} - \frac{1}{m} \sum^m (\text{neighbors of } \mathbf{v}_{i,j})\|^2 & \text{if } m \neq 4 \\ \|\mathbf{v}_{i+1,j} - 2\mathbf{v}_{i,j} + \mathbf{v}_{i-1,j}\|^2 + \|\mathbf{v}_{i,j+1} - 2\mathbf{v}_{i,j} + \mathbf{v}_{i,j-1}\|^2 & \text{if } m = 4, \end{cases} \quad (\text{Eq. 5.4})$$

where m is the valence of $\mathbf{v}_{i,j}$. For vertices along the surface boundaries (excluding corner points on boundary), we use:

$$\text{fair}(\mathbf{v}_{i,j}) := \sum_{i,j} \|\mathbf{v}_{i+1,j} - 2\mathbf{v}_{i,j} + \mathbf{v}_{i-1,j}\|^2, \quad (\text{Eq. 5.5})$$

where $\mathbf{v}_{i+1,j}$ and $\mathbf{v}_{i-1,j}$ are neighbors of $\mathbf{v}_{i,j}$ along the boundary. Note that the fairness term in Liu's method is a discrete bending energy. Minimizing this energy is equivalent to minimizing the sum of squared principal curvatures, i.e., $k_1^2 + k_2^2$, while minimizing the squared Laplacian is equivalent to minimizing the square of mean curvature, i.e., $(k_1 + k_2)^2$. These terms are close, but we found the bending energy more effective for vertices of valence four, which predominate the quad-meshes that were experimented with.

For each subgraph, say $G_s \in G$, we extract all its associated edges on the surface: $e_0, e_1, \dots, e_{n-1}$, and apply the following edge connection term to enforce compatible edge lengths, so that we can connect instances of S-quads on the KT-surface:

$$F_e := \sum_{G_s} \sum_{i=0}^{n-1} \left(\|e_i\|^2 - \|e_{i'}\|^2 \right)^2, \text{ where } i' = (i+1) \text{ modulus } n. \quad (\text{Eq. 5.6})$$

In addition to edge lengths, we further need to constrain the diagonal lengths to enforce compatible shape for instances of S-quads. Given $q_0, q_1, \dots, q_{n-1}$ as the instances of a given S-quad, we define $ac(q_i)$ and $bd(q_i)$ as the two diagonals of q_i , corresponding to the diagonals of the S-quad. The following diagonal term enforces compatible diagonal lengths among the instances:

$$F_a := \sum_{G_s} \sum_{i=0}^{n-1} \left[\left(\|ac(q_i)\|^2 - \|ac(q_{i'})\|^2 \right)^2 + \left(\|bd(q_i)\|^2 - \|bd(q_{i'})\|^2 \right)^2 \right]. \quad (\text{Eq. 5.7})$$

Using the diagonal term alone is insufficient to guarantee compatible shape among the instances of a given S-quad. Since we consider non-planar quads, which act like tetrahedrons in 3-space, instances in the same cluster could be optimized to be mirror of each other, but yet still having compatible edge and diagonal lengths. Since mirror reflection is not rigid body transform, we cannot tile them with the same S-quad. Hence, the following orientation term is introduced to enforce compatible orientation for instances in the same cluster. This is expressed by the signed volume (determinant) of 4-vertices in 3-space:

$$F_o := \sum_{G_s} \sum_{i=0}^{n-1} \left(V(q_i) - V(q_{i'}) \right)^2, \quad (\text{Eq. 5.8})$$

where $V(q_i)$ denotes the signed volume of q_i .

The following objective function summarizes all the above terms:

$$\min f := \mu_1 F_d + \mu_2 F_f + \mu_3 F_e + \mu_4 F_a + \mu_5 F_o, \quad (\text{Eq. 5.9})$$

where μ_i 's are user-specified weighs. Since we regard F_e, F_a , and F_o as hard constraints whereas F_d and F_f as soft constraints, we set in our current implementation $\mu_3 = \mu_4 = \mu_5 = 1$, and the

typical values of μ_1 and μ_2 to be 0.0001. Compared to the edge/diagonal/orientation constraints, the fairness and distance contribute very little to the objective function, and therefore, these terms can hardly converge to zero. We use the Conjugate Gradient method (which implements the Polak-Ribiere minimization) from the book *Numerical Recipe* to solve the above unconstrained non-linear programming.

5.6 Implementation Details and Results

5.6.1 Clustering Enhancement

Recall that *Minimum K-clustering Sum* is an NP-hard optimization problem [4], we use a number of heuristics to further improve and control the clustering quality: First, we allow inter-actively merging and breaking clusters. After each update, our system can amend the edge sharing graph and instantly feedback the degree of flexibility and edge connection constraints. Two examples, which were edited, are TOWER ($K = 9$) and SEASHELL ($K = 21$): the approximation errors of their resultant KT-surfaces are 1.43 and 3.55, respectively; after re-arranging the clusters, the approximation errors are reduced to 1.37 and 3.34, respectively, see Table 5.1. Another way is to enlarge the size of candidate set in hierarchical clustering. The larger the size is, the closer to an exhaustive search the process will be. In addition, we also initiate an auto-merge that examines the cluster pattern, i.e., the four subgraphs that the associated S-quad edges belong to. If two clusters have exactly the same pattern, they can be merged without reducing the degree of flexibility. This can quickly reduce K , particularly in symmetric surface regions. Lastly, we find it useful to add also a clustering configuration defined by a standard K-means clustering to enrich the candidate pool because hierarchical clustering may be locked in a local minimum.

Surface	K	D.O.F.	$\widetilde{F}_d$	$\widetilde{F}_f$	$\widetilde{F}_e$	$\widetilde{F}_a$	$\widetilde{F}_o$	Approx. Error
Tower	24	40	5.63E-04	2.43E-02	3.44E-08	1.37E-10	2.23E-07	0.05
	18	29	4.33E-03	3.94E-02	6.41E-07	3.39E-08	3.95E-05	0.36
	9	9	1.67E-02	5.61E-02	6.28E-06	1.97E-07	2.17E-04	1.37
Tunnel	12	35	1.78E-01	5.48E+00	2.63E-04	1.30E-06	1.88E-05	1.10
	10	29	9.35E-01	7.90E+00	1.37E-03	2.66E-06	4.35E-05	5.75
	8	22	2.11E+00	1.17E+01	1.84E-03	5.70E-06	5.56E-05	12.97
Sea Shell	96	156	3.91E-03	9.78E-03	2.84E-09	4.46E-09	1.99E-08	0.89
	60	71	7.03E-03	5.95E-03	2.30E-08	1.93E-08	8.96E-08	1.35
	44	52	8.16E-03	1.12E-02	3.15E-08	2.01E-08	9.81E-08	1.86
	21	27	1.47E-02	1.81E-02	5.66E-09	5.43E-09	1.79E-08	3.34
Decocube	15	27	5.52E-04	2.31E-02	6.73E-09	8.37E-12	6.74E-08	0.26
	9	13	1.22E-03	2.51E-02	8.98E-09	4.24E-11	1.75E-07	0.58
	3	5	7.13E-03	3.42E-02	7.13E-08	3.34E-10	8.17E-07	3.38
	1	3	1.85E-02	4.87E-02	3.27E-15	9.34E-14	3.54E-13	8.77
Monkey Saddle	34	67	1.99E-04	5.55E-03	9.62E-09	3.92E-08	1.09E-06	0.02
	28	47	4.47E-04	5.50E-03	1.75E-08	9.97E-08	1.04E-06	0.05
	22	27	1.05E-03	6.83E-03	9.10E-08	3.22E-07	2.91E-06	0.13
	16	19	3.58E-02	1.15E-03	3.39E-08	1.32E-07	9.06E-07	4.33
DeBeers Ginza	48	73	5.35E-04	3.56E-02	1.82E-06	4.23E-10	8.41E-08	0.00
	44	65	8.15E-04	3.56E-02	2.16E-06	8.83E-10	1.65E-07	0.01
	39	60	2.76E-03	4.20E-02	1.34E-07	8.85E-11	3.20E-09	0.03
	34	51	4.52E-03	4.94E-02	1.99E-07	1.88E-10	3.66E-09	0.04
	30	44	1.03E-02	5.79E-02	4.13E-07	1.77E-10	9.50E-08	0.09
	27	40	7.08E-02	4.47E-03	9.65E-07	1.25E-09	2.52E-06	0.65

Table 5.1: Optimization results on surfaces shown in Figure 5.6.

Figure 5.5: (a) K-set DE BEERS GINZA ($K = 27$); (b) original surface of SWISS RE TOWER; (c) K-set SWISS RE TOWER ($K = 21$). There is only a little visual degradation for smaller K .

5.6.2 Results

From Figure 5.6 and Table 5.1, the general observation is that the larger the K , the better the KT-surface approximates the original input. When K decreases, we gradually lose flexibility to retain geometric details in the KT-surface. Table 5.1 summarizes the KT-surface properties; the columns (from left to right) are: K , degree of flexibility, the terms in the objective function, and the approximation error. For fair comparison, we uniformly scale the surfaces into a unit cube and the terms in the objective function are normalized by the number of vertices, edges, or faces, accordingly. Two data models are particularly interesting. When K decreases, the opening in the SEASHELL gradually shrinks, appearing like a symmetrization process. For the MONKEY SADDLE, we gradually lose the flexibility to represent the frontal curly part as it has mirrored left and right halves and its quads are highly curved, thus limiting the clustering choices due to the orientation constraint, see below for summarized symmetry conditions:

- TUNNEL: multiple rotation symmetry about its medial axe and (one global and two local) reflection symmetry;

Figure 5.6: Resultant K-set tilable surfaces (from top to bottom): SWISS RE TOWER (528 quads); TUNNEL (1152 quads); SEASHELL (400 quads); DECOCUBE (480 quads); MONKEY SADDLE (100 quads); and DE BEERS GINZA (80 quads).

- TOWER: multiple rotation symmetry about its medial axe and approximate reflection symmetry about its middle;
- DECOCUBE: multiple mirror-reflection / rotational symmetry;
- MONKEY SADDLE: mirror-reflection (left and right) and rotational symmetry (front and back);
- SEASHELL: one mirror-reflection symmetry;
- DE BEERS GINZA: no symmetry.

For TOWER, we found from our experiment that both hierarchical and k-mean clustering can exploit approximate symmetry about its horizontal plane in the middle, see the clustering pattern shown in Figure 5.6. See also Figure 5.5 for the demonstration of KT-surfaces as the

Figure 5.7: The non-uniform arrangement of quads around the TOWER tip.

building exteriors of DE BEERS GINZA and TOWER (that mimicks Swiss Re Tower).

5.6.3 Limitations

First, the input surface is assumed to be noiseless so that the similarity metric can provide good initial clustering. Second, our optimization does not scale well. Currently we are limited to surfaces of less than 1500 quads. A possible solution is to use quad simplifications [15], and another is to first segment the surface and optimize it in parts. Third, constraining the quads to be planar is a different and harder problem. This is because it is equivalent to replacing F_o to be sum of all $V(q_i)^2$, which instead of enforcing compatible orientation for instances in the same cluster, it enforces every single instances to have zero volume (or planar). This actually further limits the degree of freedom of the solution space under the edge and diagonal constraints, compares to the original orientation constraint. Thus, it was not considered in this current work. Lastly, for many models, we may not be successful in reducing K to small number. Like in Figure 5.6, TOWER gets severe distortions already with $K = 9$ tiles. If we push it further down to $K = 2$, the TOWER degenerates and collapses due to the non-uniform arrangement of quads around the tower tip (see Figure 5.7). Also when the quads have irregular shapes, like in the TOWER, it becomes too ambiguous to cluster them. However, for some models we

succeeded to go down to $K = 1$, e.g., the DECOCUBE becomes cube-like while the MONKEY SADDLE is completely flattened.

5.7 Summary

This work solved a challenging problem in freeform surface modeling, the solution is the K -set tilable surface. It allows us to closely approximate an input surface with instances of a small set of K quads. This solution may contribute to an effective cost reduction in the physical assembly of the given surface. In detail, we introduced the edge sharing graph structure to analyze the edge connection constraints and also to learn the degree of flexibility from the clustering. We further formulate constraints according to the S-quad tilability and the surface approximation, and devise a non-linear optimization model to progressively iterate the vertices positions to achieve the construction of KT-surfaces. In the work, we push K to be particularly small. However, we believe that the advantage of the technique is for designing surfaces with rather moderate K .

Chapter 6

Tile-based Reconstruction of Interlocking Structure

6.1 Introduction

Figure 6.1: Different kinds of 3D interlocking puzzles that resemble different objects.

Puzzles have always been fascinating, intriguing and entertaining adults and kids. Naturally, several computational methods have been developed for solving or generating puzzles [23, 29, 43, 9]. 3D puzzles, in particular after assembled, usually resemble objects, for example

Figure 6.2: (a) A burr puzzle made from BIMBA; and (b) the eleven puzzle pieces after disassembly.

those interlocking puzzles shown in Figure 6.1. In this work, we are interested in the making of burr puzzles, which are particularly attractive, complex, and highly challenging to solve (Figure 6.2). A burr puzzle is a 3D model that consists of interlocking components with a single-key property [12, 13, 34]. That is, when the puzzle is assembled, all its parts are notched except one single key component which remains mobile. Unlike conventional puzzle games such as jigsaw puzzles, where the challenge mainly arises from the quantity of puzzle pieces, a burr puzzle attains a very high difficulty index with only a small number of puzzle pieces. Such difficulty index relates to the combinatorial complexity in the puzzle piece arrangement and assembling order. Moreover, due to the single-key interlocking property, burr puzzles could have more serious applications. For example, during a 3D printing process of large size 3D solid model, simply cutting the model into pieces that fit the printing volume, and glue them back after printing would result an irreversible process that cannot tolerate mistakes. Converting the 3D solid model into burr puzzle can solve the above mentioned issue easily. As long as the single key is secured, the printed model is stable without the necessary nailing, screwing or gluing operations.

6.1.1 Background on Burr Puzzles

Burr puzzles have been massively produced in Asia as early as in the 18th century, but there is no consensus as to the origin of the burr puzzle. The IBM research website [34] dated burr puzzles as early as 1803 in the Bestelmeier Toy catalogs while Coffin [10] quoted Slocum's work, which traced its history to at least 1698 in Germany. The six-piece burr puzzle can be constructed by arranging three pairs of mutually-perpendicular rods, notching each other in a central region (Figure 6.3(a)). There are only very limited number of burr puzzle variations for a long time until 1978 when Cutler [12, 13] employed computers to systematically and thoroughly analyze all possible combinations of six-piece burr puzzles. Cutler is an American mathematician who worked on analyzing burr puzzles. In addition to six-piece burrs, he also worked on various kinds of burr puzzles, such as rectilinear burrs, non-rectilinear burrs, and box-filling burrs, see [14].

Figure 6.3: (a) A traditional cuboid burr puzzle; and (b) the canonical six-piece burr puzzle.

6.1.2 Properties on Burr Puzzles

The burr puzzle pieces have specially-designed geometric structures which yield the unique characteristic of being *interlocking*: once a burr puzzle is assembled by slipping in the last puzzle piece, no other pieces can be taken out unless we first move the last piece, which is called the key. Since the key piece locks the entire 3D model, the whole geometric structure of the 3D puzzle can remain stable without glue, screw, and nail, but at the same time, we can still disassemble and then re-assemble it like common puzzles.

In this work, we take a tile-based approach to generate burr puzzles from a given 3D geometric model, in contrast to the traditional burr puzzles that are mainly cuboid in shape (Figure 6.3(a)). The result is a partition of the 3D shape into perfectly-interlocking puzzle pieces (Figure 6.2) that can be disassembled with a single moveable key piece. Our basic building block is a single canonical six-piece burr puzzle, in which we call it a *knot* for simplicity (Figure 6.3(b)). We illustrate our method by first generating a single-knot puzzle (6 puzzle pieces only) from the 3D model. Then we extend it to the more complicated multi-knot puzzle (a tiled network of knots) that contains a larger number of puzzle pieces.

Note that simply tiling the knots does not naturally preserve the single-key property of the canonical single-knot case. Ensuring such property is a highly challenging issue since there might be a large number of puzzle pieces in the geometric construction, and yet the puzzle pieces have to be fully interlocked with suitable connection types (interdependency). Moreover, the assembly/disassembly of burr puzzles requires each piece to have a non-blocking motion space so that they can be assembled/disassembled in a specific trajectory. Thus, puzzle pieces have to be carefully generated to avoid blocking the trajectory of some other puzzle pieces. To help visualizing the assembly/disassembly of our generated burr puzzle, we further develop visualization methods to automatically illustrate and annotate the puzzle assembly process.

6.2 Overview

The problem of making a burr puzzle from a 3D model is a volume partitioning problem. The given 3D model is split into disjoint components that can be assembled to form an interlocking burr puzzle. Our solution is based on extending the pieces of a known canonical burr puzzle into larger pieces that conform to the shape of the input model without violating their interlocking properties. To simplify the description of the method, we first start with a basic case of making a six-piece burr puzzle, or a single-knot burr puzzle, and then extend our solution to the general tile-based multi-knot burr puzzle.

Figure 6.4: (a) Single knot: the canonical six-piece burr; (b) embedding the single knot into the 3D model, SQUIRREL; (c) extending the “blade” to partition the volume without touching the center burr lock; and (d) pieces after flesh attachment.

To generate a single-knot burr puzzle, we embed a canonical six-piece burr (knot) in the input 3D model. The method is illustrated in Figure 6.4. The knot in Figure 6.4(a) is first partitioned into inner and outer parts. The inner part, which remains a canonical six-piece, is placed entirely inside the 3D model (Figure 6.4(b)). Then the outer pieces are extruded using an anisotropic (axial) scaling till they go beyond the 3D model. Then we can apply a CSG intersection between the extruded six pieces and the given 3D model to produce the puzzle

“skeleton” as shown in Figure 6.4(c). The last step is to compute the missing octant volumes (“fleshes”) and attach them to the “skeleton” pieces to yield the single-knot burr puzzle shown in Figure 6.4(d) (Section 6.3).

The general case of making a multi-knot burr puzzle is based on the basic case. The goal is to tile multiple knots into a network so that the resultant puzzle owns the single-key property and all puzzle pieces are interlocked but still can be disassembled. The network is in general an orthogonal cubic tile structure with each knot corresponding to a unique center point of a cubic tile. To work out the connection, we first compute the shortest path tree of the network to obtain a partial order of the knots. Then, we propose novel strategies to orient neighboring knots and modify adjacent puzzle pieces to connect them, aiming at enforcing the single-key and disassemblability properties. Finally, we compute the disassembly order of the pieces, and generate the geometry of each piece with CSG boolean operations (Section 6.4).

To help understanding how the puzzle can be disassembled/assembled or preparing an “instruction manual,” we develop a series of visualization methods in the spirit of [61] to *automatically* generate annotated disassembly/assembly animation for each created puzzle (Section 6.5). In Section 6.6, we present our results of various burr puzzles made from different 3D models, and the rapid-prototyped burr puzzles. Finally, Section 6.8 draws the conclusion.

6.3 Single-Knot Burr Puzzle

As mentioned before, we employ the canonical six-piece burr puzzle (single-knot) (Figure 6.3(b)) as the basic tile element for our burr puzzle construction with the fundamental analysis of its locking mechanics. Then we use the analysis of the orientation difference of two knots together with their connectivity and dependency, for extending it into the general multi-knot puzzle in Section 6.4.

6.3.1 Locking Mechanics

Before we go on, we first describe the locking mechanics of a knot. Without loss of generality, we denote the six puzzle pieces of a single knot as $\mathbf{A}_1$, $\mathbf{A}_2$, ..., and $\mathbf{A}_6$ (for knot ($\mathbf{A}$)) (Figure 6.3(b)), and consistently color-code each of them in order to facilitate discussion throughout this paper. The disassembly of a single knot must start with the only moveable key $\mathbf{A}_1$. $\mathbf{A}_1$ can only slide a little bit outward as it is later blocked by $\mathbf{A}_4$ and $\mathbf{A}_5$. After such little move, the burr lock becomes *partially unlatched* (namely the *activated* state), and we can then take out $\mathbf{A}_2$ or $\mathbf{A}_3$ entirely. The selection of a removable piece at each step may affect the disassembling ordering (path) of the remaining pieces. In general, as one more piece is removed, the choice of further removable pieces increases. Figure 6.5 shows all possible disassembling sequences (paths) in the form of a graph. Note that the notation $\mathbf{A}$ is dropped for clarity in the graph and the *unlocked* state means that all remaining pieces can be taken out in any order. A video sequence of assembling a knot is shown in the supplementary video.

6.3.2 Knot Embedding

To create burr puzzles from a given 3D shape, we can simply embed the knot into the shape (Figure 6.4(b)) and extend the “blades” of burr pieces along the three principle axes to partition the shape (Figure 6.4(c)). By this, we regard the problem of making a burr puzzle from a 3D model as a volume partitioning problem. The extension of “blades” can be achieved by applying an anisotropic (axial) scaling only on the blades without touching the center notch of the burr lock. Note that such extension does not violate the interlocking property since each piece is extended only radially from the knot center. After that, the six burr pieces are formed by CSG intersection with the given 3D shape (Figure 6.4(c)).

Figure 6.5: Possible disassembling orders: the vertical rectangular boxes and rounded squares next to each of them denote the states and the removable puzzle piece(s) at each state, respectively.

6.3.3 Flesh Attachment.

So far, we only form the “skeleton” of the burr puzzle. There are eight missing octant volumes (“fleshes”) as in Figure 6.4(c), and these fleshes have to be attached to the six extended puzzle pieces to complete the final puzzle. There can be many ways to attach the fleshes. The key criteria we used is to make the cutting plane of the flesh radial from the knot center to avoid potential occlusion in the assembling/disassembling trajectory of the puzzle pieces.

Here, we suggest three kinds of attachment schemes, namely *two-way*, *multi-way*, and *even* attachments. The two-way scheme attaches the eight fleshes to two out of the six pieces on the opposite side, e.g., A_1 and A_6 , A_2 and A_3 , or A_4 and A_5 (Figure 6.6 (top)). The multi-way scheme generalizes the two-way one by arbitrarily attaching a flesh to one of its three neighboring puzzle pieces, resulting in 3^8 possible choices altogether. The even attachment scheme strives for a more balanced partitioning by first partitioning each flesh into three subvolumes and then attaching each subvolume to the neighboring puzzle piece (Figure 6.6 (bottom)). The choice of flesh attachment schemes is decided by the users. In practice, we couple the flesh

Figure 6.6: (a) Two-way attachment; and (b) even attachment by three 45-degree cutting planes.

Figure 6.7: Fragments may be resulted during partitioning.

attachment step with the skeleton construction step, by subtracting unwanted subvolumes from the given object using CSG operations.

Note that a puzzle piece may result in multiple disjoint parts during the partitioning in knot embedding and/or flesh attachment (Figure 6.7). We have to constrain the partitioning process to a single connected component to avoid fragmentation during the CSG operations.

6.4 Multi-Knot Burr Puzzle

Extending the burr puzzle from single-knot to multi-knot results in more complicated and challenging 3D puzzles with a larger number of interlocking puzzle pieces. To do so, we need a series of procedures: designing the knot tile network, computing the knot interdependency and orientation to maintain the single-key property, modifying certain pieces to connect neighboring knots, and finally generating the puzzle pieces.

Figure 6.8: Constructing a multi-knot burr puzzle: (a) a given 3D model; (b) a network of knots; (c) partition the model volume for each knot; (d) compute the knot interdependency; (e) compute the knot orientation; and (f) generate the puzzle pieces.

6.4.1 Knot Network Design

Given an input 3D object, our first step is to design a tiling network of knots that fits inside the volumetric space of the object (Figures 6.8(a)-(b)). We employ an interactive GUI to load the 3D model, position the knots inside the volume, and connect them into an undirected network. Since the 6 pieces of a knot are axis-aligned (orthogonal) in nature, edges in the knot network are also axis-aligned. Therefore, the placement of neighboring knots is constrained to be axis-aligned. During the knot network design, the system renders partition planes, i.e., the mid-plane between neighboring knots (Figure 6.8(c)). This allows us to visualize the partitioned sub-volumes corresponding to each knot, thereby avoiding uneven subdivision of the object volume among the knots. The last input in this step is the specification of the *key knot*. That is the knot containing the key piece of the entire puzzle (Figure 6.8(b)).

Note that, when designing the knot network, edges are undirected (the knot dependency is not

yet decided) and the knot orientation is not yet determined. Our methods to be described later can automatically compute them.

6.4.2 Knot Orientation

Due to its axis-aligned nature, the knot network is a subgraph of a complete 3D grid graph. For any two neighboring grid points in the graph, there must be an edge connecting them to ensure the graph connectivity. When placing a knot at a grid point, we have 48 possible choices of orientations via rotation and mirror-reflection. This is due to the fact that there are 24 different orientations by applying axis-to-axis rotations; in addition, we can obtain another 24 different orientations by applying mirror reflection of them because the burr structure is asymmetrical. Note that mirror reflection of a knot is still a valid burr puzzle. To facilitate the knot connection, we further divide the 48 orientations into two groups using the following group operations: 180° -rotation about X , Y , or Z axis and mirror reflection. This gives us groups G_1 and G_2 , each with 24 orientations.

Figure 6.9: Connection types of neighboring knots.

To connect neighboring knots, we found that the orientation of two neighboring knots, say **A** and **B**, must be chosen from different orientation groups. That is, if **A** is from G_1 (or G_2), **B** must be from G_2 (or G_1). In this way, the proposed constraint can always result in a distinctive contact (90-degree-off) between all neighboring knots, as depicted by the top-left thumbnails (cross-sections) in Figure 6.9(a). This pattern can support the connection technique described in the next step.

6.4.3 Knot Connection and Dependency

With the above orientation constraint, we can connect neighboring knots and then create interdependency to maintain the single-key property. This requires modification on the puzzle pieces. To facilitate the discussion, we denote a knot as **A** and its i -th piece as $\mathbf{A}_i$. We also employ the same subscript numbering and color-coding of pieces as in Figure 6.3(b).

Connection type #1: Strong dependence. Consider two neighboring knots **A** and **B**. In this type, we orient $\mathbf{B}_1$ to face $\mathbf{A}_i$ where $\mathbf{A}_i$ can be any piece of **A** other than $\mathbf{A}_1$. Next, we cut $\mathbf{A}_i$ into two halves as in Figure 6.9(a) and merge them into $\mathbf{B}_4$ and $\mathbf{B}_5$, respectively. Since $\mathbf{B}_1$ is the key piece of knot **B** (Figure 6.5), by orienting $\mathbf{B}_1$ to face knot **A**, we block the movement of $\mathbf{B}_1$ and make **B** fully dependent on **A**. That is, **B** cannot be unlocked without first unlocking **A**. We denote such strong dependence by $\mathbf{B} \rightarrow \mathbf{A}$.

Connection type #2: Independence. Two neighboring knots can be connected but not dependent on each other in the assembly/disassembly order. This connection type is called independence. One example is shown by the green edge in Figures 6.8(d)&(e). Geometrically, it is a variant of Type #1, and achieved by orienting **B** so that both $\mathbf{B}_1$ and $\mathbf{B}_3$ face away from **A**, and then by modifying the puzzle pieces as in Type #1 (the result is shown in Figure 6.9(c)). With this arrangement, both **A** and **B** can be unlocked independently, even though they are connected. We denote this connection type as $\mathbf{B} - \mathbf{A}$.

Connection type #3: Partial dependence. Figure 6.9(b) demonstrates how pieces are modified to construct this type. First, we orient $\mathbf{A}_1$ and $\mathbf{A}_6$ to block $\mathbf{B}_1$. Then, we modify these three pieces as in Figure 6.9(b) (lower right) so that $\mathbf{B}_1$ is blocked only by $\mathbf{A}_1$. The introduced gap between modified $\mathbf{B}_1$ and $\mathbf{A}_1$ & $\mathbf{A}_6$ is just large enough to allow $\mathbf{B}_1$ to activate $\mathbf{B}$ (the little forward move) after $\mathbf{A}_1$ has moved outwards (downwards in the figure). Recall that a knot is activated by slightly moving outwards its key piece (Figure 6.5). So when $\mathbf{A}$ is locked, so as $\mathbf{B}$. But when $\mathbf{A}$ is activated, so as $\mathbf{B}$, even though other pieces of $\mathbf{A}$ have not been moved yet. In other words, $\mathbf{B}$ can still be unlocked as long as $\mathbf{A}$ can be activated. This connection type is useful to provide flexibility when the knot network is heavily interlocked. We denote this type as $\mathbf{B} \rightarrow \mathbf{A}$.

Connection type #4: Partial dependence. The last connection type is a variant of Type #3, in which we orient $\mathbf{A}_2$ and $\mathbf{A}_3$ instead of $\mathbf{A}_1$ and $\mathbf{A}_6$ to contact $\mathbf{B}_1$ (Figure 6.9(d)). In this way, $\mathbf{B}$ can be activated only if $\mathbf{A}_3$ moves out of its place. Note that $\mathbf{A}_3$ is in a later stage of the disassembly compared to $\mathbf{A}_1$, thus making it a more secure type of partial dependence as compared to Type #3. We denote it as $\mathbf{B} \rightarrow \mathbf{A}$.

6.4.4 Determining Knot Orientations and Connections

Given a knot network, $\mathbf{G} = (\mathbf{V}, \mathbf{E})$, where $\mathbf{V} = K, A, B, \dots$ is the set of knots and $\mathbf{E}$ is the set of undirected edges, we now have to determine the dependency among knots and the orientation of each knot. Our goal is to find a set of dependency type for each edge, $Type(\mathbf{E})$ and the orientation of each knot, $R(\mathbf{V})$, such that the resulting puzzle skeleton maintains the single-key interlocking property. Mathematically, we are seeking a non-unique mapping function, $f : \mathbf{G} = (\mathbf{V}, \mathbf{E}) \rightarrow \mathbf{G}' = (R(\mathbf{V}), Type(\mathbf{E}))$, which leads $\mathbf{G}'$ to a valid single-key interlocking burr puzzle. Our method performs the following steps to solving the mapping function:

Step 1: Compute a shortest path tree. With the user specified key knot $\mathbf{K}$, we apply the Dijkstra's algorithm starting from $\mathbf{K}$ to compute a shortest path tree, $\mathcal{T}$, with $\mathbf{K}$ as the root

Figure 6.10: Illustrating knot orientations and connections determination using the ISIDORE HORSE as an example: *Step 1: Compute a shortest path tree:* (a) Note that red circled knot indicates the user specified key $\mathbf{K}$ and is the root node of the shortest path tree, and blue lines indicates the edges of the tree. *Step 2: Candidate knot orientations:* (b) $\mathbf{K}$ is rotated to face to the back of the ISIDORE HORSE, so that $\mathbf{K}_1$ does not face to any neighbouring knots; (c) other knots are rotated so that their key pieces are facing to their corresponding parent knots. *Step 3: Candidate connection types:* (d) Type #1 is being chosen for each pair of neighboring knots. *Step 4: Determining orientation and connection:* after simulating the disassembly starting from $\mathbf{K}$, all pieces can be disassembled from the initial orientation and connection. Note that all $\mathbf{B}_3$ and $\mathbf{B}_5$ are facing away from neighboring knots.

of $\mathcal{T}$. Using the 4 knots placement in ISIDORE HORSE as an example (Figure 6.10(a)), the resulting shortest path tree is a linear list rooted at $\mathbf{K}$.

Step 2: Candidate knot orientations. Then, we can rotate $\mathbf{K}$ so that its key piece $\mathbf{K}_1$ does not face any neighboring knots, thereby ensuring $\mathbf{K}_1$ to be always movable when the puzzle disassembly starts. Thus, $\mathbf{K}$ is being rotated to face the back of the ISIDORE HORSE as shown in Figure 6.10(b). Based on the parent-child relationship in $\mathcal{T}$, we can determine a disassembly order, and hence the interdependency constraint, for each pair of neighboring knots. If $\mathbf{A}$ and $\mathbf{B}$ are a pair of parent and child knots in $\mathcal{T}$, we arrange $\mathbf{B}_1$ to face $\mathbf{A}$ so that $\mathbf{B}$ is dependent on $\mathbf{A}$, either fully or partially. Note that, at this moment, the orientation of $\mathbf{B}$ can still have four possible choices (by a 180° rotation along $\mathbf{B}$ to $\mathbf{A}$ and/or mirror reflection about plane of $\mathbf{B}_1$) in an orientation group that is different from that of $\mathbf{A}$. Figure 6.10(c) shows how the four knots orientated after this step.

Step 3: Candidate connection types. In addition to knot orientation, we determine a set of candidate connection types for each pair of neighboring knots. If the pair is associated with an edge in $\mathcal{T}$, we must connect the two knots with Type #1, #3 or #4, so that $\mathbf{B}$ can always depend on $\mathbf{A}$ to maintain the interlocking (see Figure 6.10(d)). On the other hand, if the pair is not associated with any edge in $\mathcal{T}$, we use the independence connection Type #2 to connect them so that the resultant puzzle can be more strongly connected.

Step 4: Determining orientation and connection. With the candidate orientations and connection types over the knot network, we can simulate the disassembly starting from knot $\mathbf{K}$ with its key piece. The goal is to determine the orientation and connection for all knots so that all pieces can be disassembled in order during the simulation. Note that we can follow the logics in Figure 6.5 to compute the valid puzzle piece moves locally in each knot together with the logics previously defined for the four connection types. Given a puzzle pieces $\mathbf{A}_i$ and its corresponding disassemble motion vector, $\vec{d}$ (from the logics in Figure 6.5), our simulation evaluate the amount of collision volume, $Vol(\mathbf{A}_i, \vec{d})$, that $\mathbf{A}_i$ is producing. If the objective function,

$$Err(\mathbf{G}') = \sum_{\forall \mathbf{A}_i} Vol(\mathbf{A}_i, \vec{d}) \neq 0, \quad (\text{Eq. 6.1})$$

our simulation identifies the corresponding orientation (or $R(\mathbf{V})$) and connection type (or $Type(\mathbf{E})$) are invalid.

In our current implementation, we use a greedy algorithm to make choices during the simulation. That is, we attempt to arrange $\mathbf{B}_3$ and also $\mathbf{B}_5$ to face away from neighboring knots of $\mathbf{B}$ when choosing the knot orientation of $\mathbf{B}$ because $\mathbf{B}_3$ is a crucial milestone piece in the disassembly (Figure 6.5). In addition, we prefer Type #1 over #3 and #4 as the dependence connection because it has the strongest dependency among the three. Note that the solution given by these 4 steps does not necessarily be unique, user can always override the decision of greedily choosing Type #1 over #3 and #4, whenever one or more choices are available.

6.4.5 Generating Puzzle Pieces

After determining the knot orientations and connections, we can generate the “skeleton” of multi-knot burr puzzles as in the single-knot case. At the same time, we can also modify the puzzle pieces according to the determined connection type. Then, starting from the key piece, we can attach the “flesh” into the skeleton puzzle pieces by computing the motion space of each puzzle piece while ensuring a non-blocking trajectory. Our implementation employs the CGAL library API to generate an action list, and performs the CSG boolean operations in 3D Studio Max via scripting. An action list encodes a sequence of CSG operations for constructing the geometry of each puzzle piece. Each action in the list is basically a CSG operation such as $A(\cap/\cup/-)B = C$, where A , B , and C are mesh models.

6.4.6 Extensions

Our multi-knot puzzle modeling framework can be further extended in the following two ways:

Non-Orthogonal Connection. So far, neighboring knots are arranged exactly axis-aligned. We can generalize orthogonal knot connection to be non-orthogonal by shearing the volumetric space in-between neighboring knots (Figure 6.11 (a)). Since such shearing does not block the motion trajectory of puzzle pieces in the assembly, we can retain the interlocking and connection structure among the puzzle pieces. To construct the non-orthogonal connection as shown in Figure 6.11 (a), we first arrange three knots in the DRAGON body in zigzag way (with 55° turning angle). Then, we can apply a piecewise shearing along one specific direction (while isometric along the others) to deform the subspace in the puzzle skeleton structure. Finally, we can continue with the standard CSG operations in our multi-knot framework to generate the puzzle pieces.

Disjoint Knot Networks. Another interesting variation is that we can produce disjoint knot networks within a 3D model, and yet we can connect them by merging puzzle pieces among

Figure 6.11: (a) Non-orthogonal connection; and (b) connecting disjoint knot networks.

them. As shown in Figure 6.11 (b), we in fact start with three disjoint knots here. By merging relevant contacting pieces (those become grey) between the middle and left knots (as well as the middle and right knots), we can connect these three disjoint knots. However, it is worth to note that such connection does not enforce the single-key property. In this example, we can add in two pairs of tabs and blanks on the blue and the two red pieces (left and right) as indicated in the figure. The tabs on the blue piece can block the motion volume of the two red pieces, which are the keys of the left and right knots. As a result, these two knots become dependent on the blue piece of the middle knot, hence leaving a single key in the entire system. The general rule is that we have to pick one of the knot networks as the key, and block the key pieces of all the other disjoint knot networks by pieces subsequently dependent on the key network.

Figure 6.12: Illustrating the puzzle disassembly: (a) motion arrow placement to indicate puzzle piece movement; (b) number label to show the movement stage; (c) contact areas coding with zoom-in view to illustrate the interlocking structure; (d) successive views showing the camera movement to expose the next puzzle piece in-motion; and (e) annotating multiple puzzle pieces in motion.

6.5 Assembly/Disassembly Illustration

Assembling burr puzzles can be very challenging, particularly for multi-knot burr puzzles. To aid the understanding on the puzzle assembly/disassembly process, we develop illustrative visualization methods to automatically generate animations and figures to illustrate and annotate the assembly/disassembly process. Related literatures on illustration generation include: Mitra et al. [61] used cap, side and translational arrows to illustrate the spatial movement of mechanical parts. Agrawala et al. [1] used motion arrows as a guidance. Our method follows the principle of these works to illustrate the spatial configuration and object motion of our puzzle pieces. In particular, simultaneous movement of multiple puzzle pieces is also possible in our illustration.

6.5.1 Camera Movement

To ensure the visibility of puzzle piece in-motion, we change the camera view on the puzzle pieces. Given an initial camera position and the puzzle assembly order, we compute the visibility of each puzzle piece in-motion by rendering it together with all the other pieces in the camera view. If the proportion of its occlusion in the view is higher than a threshold ($\tau_o = 50\%$), we move our camera to a new orientation such that the camera viewing direction and the piece movement direction is within a prescribed threshold ($\tau_a = 60$ degrees). The upward vector of the camera frame remains constant to minimize the annoying tilting. For the case of multi-knot puzzles, we also automatically translate the camera to center our view on the next knot to be assembled or disassembled.

6.5.2 Motion Arrows and Numbering

Motion arrows have been used in many conventional composition diagrams in instruction manuals to indicate how parts should be assembled. With our previous puzzle formation result, we can determine the assembly/disassembly sequence, and thus, produce the visualization animation. Our method first analyses the puzzle piece movement during the assembly/disassembly, and then automatically places the motion arrows for illustration. To place a motion arrow on the moving piece, we compute the centroid of the largest visible region occupied by this piece in image space, and then project this point onto the puzzle piece surface in object space. The arrow tail is then anchored at this 3D point and its head is oriented towards the moving direction of the puzzle piece. The arrow is further translated towards the camera in order to avoid intersection with the puzzle piece (Figure 6.12(a)) and scaled properly relative to the size of the related puzzle piece.

Numbering is another common feature in many conventional illustrations. We automatically generate number labels to show the assembly/disassembly steps. We place the number labels next to the arrow head (Figure 6.12(b)).

6.5.3 NPR Stylization and Color-Coding

To better convey the geometric structure, we employ non-photorealistic rendering style similar to [47]. We color-code the canonical six pieces with RGBYMC colors, and for multi-knot puzzles, we highlight those merged pieces (modified after knot connection) with a light grey color, see Figures 6.12(a)-(e).

Furthermore, to help visualizing how burr pieces touch each other, we color-code the contact area at the innermost burr lock region. The color applied to each contact area is the color of the contacting piece (Figure 6.12(c)). Such visualization is applied only to the puzzle piece in motion, and we display also a zoom window to expose this contact area information.

6.6 Result

We applied our computer-aided-geometric design system to make a number of burr puzzles from a variety of 3D models using different knot network arrangements. Figure 6.14 shows all the 3D puzzles created from our modeling system (see the mini-map indices for the name of the puzzles). Regardless of the knot arrangements and the number of puzzle pieces contained, all the presented puzzle models can enforce the single-key property, and thus can be perfectly-interlocking upon the puzzle assembly. Note also that the small gap in the middle of CUBICBOX is a result of Type #3 connection; such a gap is introduced to leave a movement space for taking out the related key piece. Figure 6.13 depicts the corresponding knot network arrangement for each of these puzzles, using blue arrow, green edge, yellow arrow, and magenta arrow to indicate Type #1, #2, #3, and #4 connections, respectively; in addition, the key knots are colored in orange. Note that since knots connected by Type #2 are independent of each other, thereby no arrow heads are drawn. In addition, 2-TORUS is created by connecting disjoint knot networks. Its knots are not connected using any of the standard connection types.

Table 6.1 provides the detail of the 3D puzzles in three sections. The first section refers to the 3D puzzles created using the extended techniques: DRAGON and 2-TORUS. The second section refers to those produced with a rapid-prototyping counterpart, whereas the last section is for the rest. The last column in the table shows the time taken for our system to compute the knot orientations and connections, as well as to generate the puzzle pieces. These experimental results also demonstrate the possibility of using very different arrangements when designing the knot network.

Figure 6.13: Knot networks employed to construct the 3D burr puzzles presented in Figure 6.14.

Figure 6.14: Single-knot and multi-knot burr puzzles created from assorted 3D models: (a) CUBICBOX, (b) BUNNY, (c) 2-TORUS, (d) MOAI, (e) DRAGON, (f) LAURANA, (g) SHARK, (h) TORUS, (i) SQUIRREL, (j) MEGABOX, (k) ISIDORE HORSE, (l) BIMBA, and (m) CHESS.

Table 6.1: Information about the burr puzzles in Figure 6.14.

Puzzle model	#knots	#pieces	Timing (sec.)
DRAGON	3 (non-ortho.)	16	175
2-TORUS	3 (disjoint knots)	16	95
CHESS	1 (interchangeable)	7	107
MOAI	2 (evenly attachment)	11	141
TORUS	4 (2×2)	20	113
SQUIRREL	1 (single-knot)	6	73
BIMBA	2 (linear)	11	92
BUNNY	3 (L -shaped)	16	163
SHARK	3 (linear)	16	151
ISIDORE HORSE	4 (L -shaped)	21	187
LAURANA	4 (T -shaped)	21	171
CUBICBOX	8 ($2 \times 2 \times 2$)	36	196
MEGABOX	64 ($4 \times 4 \times 4$)	240	1405

6.6.1 Rapid-prototyping Results.

Figure 6.15 shows the three rapid-prototyping burr puzzles we have. They are created from 3D plastic parts printers (SOMOS and SLA machines) with different slot widths, i.e., the size of the slot in the inner part of the burr lock. The slot widths for CHESS, MOAI, and TORUS are 2mm, 2.5mm, and 3mm, respectively, and such value is controllable in the puzzle piece generation.

The CHESS model is particularly interesting because it demonstrates a unique characteristic of using the burr structure to connect and compose 3D models: “*interchangeable*.” This model consists of seven puzzle pieces with two of them originated from exactly the same skeleton puzzle piece out of the six pieces in the canonical set. The headpieces for the knight and bishop are interchangeable, see again Figure 6.15 (right). In addition, it is worth highlighting that the rapid-prototyped TORUS puzzle is a 4-knot model with 20 pieces; these pieces are perfectly interlocking with only a single key.

6.7 Implementation Issues and Limitations

6.7.1 Implementation Issues.

The automatic visualization engine was implemented by using offscreen rendering with OpenGL to obtain the depth buffer view at different time frame. Hence, we can obtain scene conditions at the viewpoint and automatically generate an animation script for 3D Studio Max to position and move the camera, to arrange the motion arrows (through scripting in 3D Studio Max), as well as to layout other illustrative elements shown in the assembly/disassembly video, see also Figure 6.16 for the assembly sequence of the BUNNY puzzle. Compared to the case of disassembly, these motion arrows are reversed. In addition, we also developed a lightweight program to help preview the puzzle disassembly/assembly sequence in a generated animation

script; see the supplementary material for this lightweight program and the simplified version of the animation scripts.

6.7.2 Limitations.

We cannot create burr puzzles from 3D shapes with parts that are too narrow or flat, and the key must be on the outer side of the knot network. Although in Section 6.4 we demonstrate how to extend the connection strategies to more complicated geometric shapes by attempting to relax the requirement of orthogonal structure, we must point out that such an extension technique is not general. The essential limitation of the orthogonal structure, in our mind, lies in the intrinsic structure of burr. Moreover, there are also no general methodologies that can always ensure that a single key locking a disjoint knot network.

More generally, a burr puzzle should better be made up of pieces that are roughly equal in

Figure 6.15: Rapid prototyping models of our 3D burr puzzles.

Figure 6.16: An assembly sequence of the BUNNY puzzle.

size, similar in shape, as well as solid and strong. In addition, input 3D models that are more symmetric are likely to produce more balanced puzzle pieces. However, these observations are also not absolute. Building burr puzzles from asymmetrical models may also be nicely shaped, for example, for artistic purpose.

6.8 Summary

This work generalizes the well-known 6-piece orthogonal burr puzzle to multi-knot burr puzzles and introduces a cubic tile network based geometric modeling system to design and create burr puzzles from 3D models. Our tile-based modeling techniques can enforce the single-key property while connecting and interlocking the puzzle pieces partitioned from the 3D models, meaning that a 3D puzzle model can be locked by a single key piece regardless of the number of puzzle pieces it consists of. Hence, the resultant 3D puzzles can be glue-less, screw-less, and nail-less, as well as interchangeable, thus allowing us to replace or reconfigure parts in the puzzle.

Key techniques proposed include the formulation of knot network to model multi-knot burr puzzles, geometric modification methods to connect knots, the computational method for knot orientation and connection, and the volumetric partitioning schemes to shape the puzzle pieces from “skeleton” and “flesh,” as well as the visualization methods to illustrate the puzzle assembly or disassembly process. All these enable us to make the 3D puzzle models perfectly-interlocking with only a single key, as demonstrated in a variety of examples we presented.

Chapter 7

Conclusion and Future Work

7.1 Conclusion

This thesis reported all the researches that we have conducted on tile-based modeling for solving various Computer Graphics problems that can be divided as followings:

- (i) **Visualizing Impossible Figure.** We reported our analysis on the spatial tiles that construct an impossible figure, which led to our solution to 3D reconstruct the geometry representation of a 2D tile-based impossible figure. We revisited the notion of impossible figure, and suggested three guiding principles for seamless 3D visualization and interaction experience of impossible figures.

With this novel 3D geometry representation and guiding principles, we proposed dynamically reconfigurable modeling based on the user's view and location to enable 3D navigation on the impossible figure geometry. The method is then extended and adopted to various applications, including 3D gaming, haptics and VR experiences. We believed that this work would give a clear guideline to the gaming community to increase the potential usage of impossible figure in their works.

- (ii) **Freeform Surface Modeling.** We reported our analysis on the edge-connectivity in quad-tiles, which grounded our proposed computational framework on solving the K-set

tilable surface problem. The clustering step in the framework is intrinsically similar to the NP-hard optimization problem - *Minimum K-clustering Sum*, we proposed a heuristic clustering approach to obtain a feasible result. By applying this work in architecture, architect can easily approximated an architectural surface using only a small set of K quads for a significant reduction of the construction cost without simplifying the surface design.

This work has inspired other research works that take the spirit of 2D/3D tiling for both 3D surface/solid modeling, for example:

- Zimmer et al. [102] proposed a rationalization method for triangle-based point-folding structures that finds a minimal set of molds for the production process.
- Song et al. [82] developed an interactive computational tool for designing reciprocal frame by employing a conformal mapping to lift the 2D tessellation over a 3D guiding surface.
- Huard et al. [33] suggested solutions to design highly repetitive freeform surface using planar polygons.

(iii) **Solid Model Partition And Redesign.** We presented our analysis on burr puzzle and different ways to orient, position, and connect two neighbouring burr puzzle. Making use of the single-key interlocking property of burr puzzle, we proposed to solve the 3D solid model decomposition problem by tiling a knot all over the 3D solid space. Using the presented method, we can now decompose a 3D solid model into pieces, and assembling them without the need of using glues, screws, and nail. For models with highly similar local features, the component pieces are even interchangeable. We also automatically generated a visualization sequence for aiding the user on assembly and disassembly. We proved the feasibility of the approach by 3D printing the rapid prototype models.

This work satisfied the spirit of 3D tiling, and inspired other works on solid model redesign and partitioning for recreational graphics and fabrication using different approaches. Luo et al. [54] designed an automatic system to partition 3D solid model into printable and assemblable pieces that fit the working volume of the 3D printer. Igarashi et al. [35] developed an interactive system to convert a solid 3D model into beadwork design with a step-by-step assembly sequence visualization. Song et al. [81] proposed another kind of partition method inspired by recursive interlocking puzzle. Ono et al. [65] presented an automatic approach on LEGO-izing 3D polygon model. Yao et al. [99] reduced the cost of 3D printing by packing and partitioning the 3D model optimistically using a level-set-based approach.

7.2 Future Work

There are multiple directions of future works. We will briefly describe each of them and discuss the possible approaches.

Visualizing the entire 3D impossible figure geometry. Our approach fails when the user is able to view a very large portion of the 3D impossible figure geometry. One possible extension is to deform the 3D impossible figure geometry on-the-fly so that i) the gaps do not exist visually in screen space, and ii) the 3D geometry dimensions still appear to be visually consistent with the impossible figure. The solution is challenging as it requires a careful formulation on the deformation optimization subjected to user's view and location.

K-set tilable general surface. We only demonstrated quad-based KT surface, which cannot fully satisfy the actual industrial need. Our approach can be extended to support surfaces that contain a mixture of different shapes, by reformulating the optimization cost functions for a more general usage.

Generating burr puzzle using different knot structure. We exclusively extend the canonical 6-pieces burr puzzle to generate different burr puzzles from 3D models. It is possible to study other burr puzzles with different interlocking mechanism for producing different types of burr puzzles.

References

- [1] Maneesh Agrawala, Doantam Phan, Julie Heiser, John Haymaker, Jeff Klingner, Pat Hanrahan, and Barbara Tversky. Designing effective step-by-step assembly instructions. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22:828–837, 2003.
- [2] Alexander Bruce. Antichamber, 2013. <http://www.antichamber-game.com/>.
- [3] Vlad Alexeev. Impossible world - impossible figures in computer games, 2001. Website: <http://im-possible.info/english/art/games/index.html>.
- [4] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Pro-
tasi. *Complexity and Approximation*. Springer, 1999. ISBN 3-540-65431-3.
- [5] Alan Blanc, Michael McEvoy, and Roger Plank, editors. *Architecture and construction
in steel*. Taylor & Francis, 1993. 640 pages, ISBN 978-0419176602.
- [6] Blizzard Entertainment, Inc. Diablo II, 2000. <http://www.blizzard.com/diablo2/>.
- [7] Alexander I. Bobenko and Yuri B. Suris. *Discrete Differential Geometry*. American
Math. Society, 2008. ISBN 978-0821847008.
- [8] Tao Chen, Zhe Zhu, Ariel Shamir, Shi-Min Hu, and Daniel Cohen-Or. 3-sweep: Extract-
ing editable objects from a single photo. *ACM Transactions on Graphics (SIGGRAPH
Asia 2013)*, 32(6):195:1–195:10, 2013.
- [9] Taeg Sang Cho, Shai Avidan, and William T. Freeman. A probabilistic image jigsaw
puzzle solver. In *IEEE Conference on Computer Vision and Pattern Recognition (CVPR)
2010*, pages 183 –190, 2010.
- [10] Stewart T. Coffin. *Geometric Puzzle Design*. A. K. Peters, 2007.

REFERENCES

- [11] Michael F. Cohen, Jonathan Shade, Stefan Hiller, and Oliver Deussen. Wang tiles for image and texture generation. *ACM Transactions on Graphics (SIGGRAPH 2003)*, 22(3):287–294, 2003.
- [12] William H. Cutler. The six-piece burr. *Journal of Recreational Mathematics*, 10(4):241–250, 1978.
- [13] William H. Cutler. A computer analysis of all 6-piece burrs, 1994. Self published.
- [14] William H. Cutler. Bill Cutler Puzzles, Inc. puzzle works, 2000. <http://home.comcast.net/~billcutler/index.html>.
- [15] Joel Daniels, Cláudio T. Silva, Jason Shepherd, and Elaine Cohen. Quadrilateral mesh simplification. *ACM Trans. Graph. (SIGGRAPH ASIA 2008)*, 27(5), 2008.
- [16] Jiri Danihelka, Lukas Kencl, and Jiri Zara. Stateless generation of distributed virtual worlds. *Computers & Graphics*, 44:33–44, 2014.
- [17] Fred D. Davis. Perceived usefulness, perceived ease of use, and user acceptance of information technology. *MIS Quarterly*, 13(3):319–340, 1989.
- [18] M. Do Carmo. *Differential Geometry of Curves and Surfaces*. Prentice-Hall, 1976. ISBN 978-0132125895.
- [19] Jean-Denis Durou, Maurizio Falcone, and Manuela Sagona. Numerical methods for shape-from-shading: A new survey with benchmarks. *Computer Vision and Image Understanding*, 109(1):22–43, 2008.
- [20] Michael Eigensatz, Martin Kilian, Alexander Schiftner, Niloy Mitra, Helmut Pottmann, and Mark Pauly. Paneling architectural freeform surfaces. *ACM Tran. Graphics (Proc. of SIGGRAPH)*, 29(3), 2010.
- [21] Gershon Elber. Modeling (seemingly) impossible models. *Computers and Graphics*, 35(3):632–638, 2011.
- [22] Bruno Ernst. *Adventures with Impossible Figures*. Tarquin, Stradbroke, England, 1987.
- [23] H. Freeman and L. Garder. Apictorial jigsaw puzzles: The computer solution of a problem in pattern recognition. *IEEE Transactions on Electronic Computers*, EC-13(2):118–127, 1964.

REFERENCES

- [24] Chi-Wing Fu and Man-Kang Leung. Texture tiling on arbitrary topological surfaces using Wang tiles. In *Proc. of Eurographics Symposium on Rendering 2005*, pages 99–104, Jun. 2005.
- [25] Jun Fujiki and Shigeru Owada. theRelativity. In *ACM SIGGRAPH ASIA Art Gallery: Emerging Technologies*, page 15, 2008. demo movie: <http://jun-fujiki.com/theRelativity/movie/theRelativity.mpg>.
- [26] Jun Fujiki and Shigeru Owada. theRelativity+inPossible, 2008. <http://jun-fujiki.com/theRelativity/index.html>.
- [27] Ran Gal, Olga Sorkine, Niloy J. Mitra, and Daniel Cohen-Or. iWIRES: an analyze-and-edit approach to shape manipulation. *ACM Transactions on Graphics (SIGGRAPH 2009)*, 28(3), 2009.
- [28] Game Yaruoze, SCE Studios Japan, Sony Computer Entertainment. Echochrome, 2008. <http://www.jp.playstation.com/scej/title/mugen/>.
- [29] David Goldberg, Christopher Malon, and Marshall Bern. A global approach to automatic solution of jigsaw puzzles. In *Proceedings of the Eighteenth Annual Symposium on Computational Geometry*, pages 82–87, New York, NY, USA, 2002. ACM.
- [30] Branko Grünbaum and G C Shephard. *Tilings and patterns*. W. H. Freeman & Co., 1986.
- [31] Alejo Hausner. Simulating decorative mosaics. In *Proc. of ACM SIGGRAPH 2001*, Annual Conf. Series, ACM, pages 573–580, 2001.
- [32] Berthold K. P. Horn. Shape from shading. chapter Obtaining Shape from Shading Information, pages 123–171. MIT Press, Cambridge, MA, USA, 1989.
- [33] Mathieu Huard, Michael Eigensatz, and Philippe Bompas. Planar panelization with extreme repetition. In *Advances in Architectural Geometry 2014*, pages 259–279. Springer International Publishing, 2015.
- [34] IBM Research. The burr puzzles site, 1997. <http://www.research.ibm.com/BurrPuzzles/>.

REFERENCES

- [35] Yuki Igarashi, Takeo Igarashi, and Jun Mitani. Beady: Interactive beadwork design and construction. *ACM Transactions on Graphics (SIGGRAPH 2012)*, 31(4):49:1–49:9, 2012.
- [36] Slavik Jablan. *Modularity in Art*. Online article: <http://www.mi.sanu.ac.rs/~jablans/d3.htm>, 1997.
- [37] Craig S. Kaplan and David H. Salesin. Escherization. In *Proc. of ACM SIGGRAPH 2000*, Annual Conf. Series, ACM, pages 499–510, 2000.
- [38] Craig S. Kaplan and David H. Salesin. Dihedral Escherization. In *Proc. of Graphics Interface 2004*, pages 255–262, May 2004.
- [39] Chih W. Khoh and Peter Kovesi. Animating impossible objects, 1999. www.csse.uwa.edu.au/~pk/Impossible/impossible.html.
- [40] M. Kilian, S. Flöry, Z. Chen, N. J. Mitra, A. Sheffer, and H. Pottmann. Developable surfaces with curved creases. In *Advances in Architectural Geometry*, pages 33–36, 2008.
- [41] Martin Kilian, Simon Flöry, Niloy J. Mitra, and Helmut Pottmann. Curved folding. *ACM Transactions on Graphics (SIGGRAPH 2008)*, 27(3), 2008.
- [42] Junhwan Kim and Fabio Pellacini. Jigsaw image mosaics. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 21(3):657–664, 2002.
- [43] Weixin Kong and Benjamin. B. Kimia. On solving 2D and 3D puzzles using curve matching. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR) 2001.*, 2001.
- [44] Johannes Kopf, Daniel Cohen-Or, Oliver Deussen, and Dani Lischinski. Recursive Wang tiles for real-time blue noise. *ACM Transactions on Graphics (SIGGRAPH 2006)*, 25(3):509–518, 2006.
- [45] A. Lagae and P. Dutré. A procedural object distribution function. *ACM Transactions on Graphics*, 24(4):1442–1461, Oct 2005.
- [46] Hans Van Lemmen. *Tiles: 1000 Years of Architectural Decoration*. Harry N. Abrams, Inc., 1993.

REFERENCES

- [47] Wilmot Li, Maneesh Agrawala, Brian Curless, and David Salesin. Automated generation of interactive 3D exploded view diagrams. *ACM Trans. Graph.*, 27:101:1–101:7, August 2008.
- [48] Xian-Ying Li, Chao-Hui Shen, Shi-Sheng Huang, Tao Ju, and Shi-Min Hu. Popup: Automatic paper architectures from 3D models. *ACM Transactions on Graphics (SIGGRAPH 2010)*, 29(3), 2010. Article 111.
- [49] Lilly Library. Swedish designer Oscar Reutersvrd’s impossible images, 2014. <http://libraries.iub.edu/swedish-designer-oscar-reutersv%C3%A4rds-impossible-images>.
- [50] Markus Lipp, Peter Wonka, and Michael Wimmer. Interactive visual editing of grammars for procedural architecture. *ACM Transactions on Graphics (SIGGRAPH)*, 27(3):102:1–10, 2008.
- [51] Yang Liu, Helmut Pottmann, Johannes Wallner, Yong-Liang Yang, and Wenping Wang. Geometric modeling with conical meshes and developable surfaces. *ACM Trans. Graph. (SIGGRAPH 2006)*, 25(3):681–689, 2006.
- [52] Kui-Yip Lo, Chi-Wing Fu, and Hongwei Li. 3D Polyomino puzzle. *ACM Transactions on Graphics (Proc. of SIGGRAPH Asia)*, 28(5), 2009. Article 157.
- [53] Louis (t). Adynatopia, 2012. <http://www.kongregate.com/games/iaopsw/adynatopia>.
- [54] Linjie Luo, Ilya Baran, Szymon Rusinkiewicz, and Wojciech Matusik. Chopper: Partitioning models into 3d-printable parts. *ACM Transactions on Graphics (SIGGRAPH Asia 2012)*, 31(6):129:1–129:9, 2012.
- [55] Fady Massarwi, Craig Gotsman, and Gershon Elber. Papercraft models using generalized cylinders. In *Pacific Graphics 2007*, 2007.
- [56] M.C. Escher Foundation. The official M.C. Escher website, 2005. Available at <http://www.mcescher.com>.
- [57] Jun Mitani and Hiromasa Suzuki. Making papercraft toys from meshes using strip-based approximate unfolding. *ACM Transactions on Graphics (Proc. SIGGRAPH)*, 23(3):259–263, 2004.

- [58] N. J. Mitra, L. Guibas, and M. Pauly. Partial and approximate symmetry detection for 3D geometry. *ACM Tran. Graphics (SIGGRAPH 2006)*, 25(3):560–568, 2006.
- [59] N. J. Mitra, L. Guibas, and M. Pauly. Symmetrization. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007.
- [60] Niloy J. Mitra and Mark Pauly. Shadow art. *ACM Transactions on Graphics (SIGGRAPH Asia 2009)*, 28(5), 2009. Article 156.
- [61] Niloy J. Mitra, Yong-Liang Yang, Dong-Ming Yan, Wilmot Li, and Maneesh Agrawala. Illustrating how mechanical assemblies work. *ACM Transactions on Graphics (Proc. of SIGGRAPH)*, 29(3), 2010. Article 58.
- [62] Yuki Mori and Takeo Igarashi. Plushie: an interactive design system for plush toys. *ACM Trans. on Graphics*, 26(3):45, 2007.
- [63] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Transactions on Graphics (SIGGRAPH)*, 25(3):614–623, 2006.
- [64] Christopher Nolan. Inception, 2010. <http://www.imdb.com/title/tt1375666/>.
- [65] Sumiaki Ono, Alexis André, Youngha Chang, and Masayuki Nakajima. [paper] lego builder: Automatic generation of lego assembly manual from 3d polygon model. *ITE Transactions on Media Technology and Applications*, 1(4):354–360, 2013.
- [66] Victor Ostromoukhov, Charles Donohue, and Pierre-Marc Jodoin. Fast hierarchical importance sampling with blue noise properties. *ACM Transactions on Graphics*, 23(3):488–495, 2004. Proc. SIGGRAPH 2004.
- [67] Martin R. Oswald, Claudia Nieuwenhuis, and Daniel Cremers. A review of geometry recovery from a single image focusing on curved object reconstruction. In *In Proceedings of the 2011 Conference on Innovations for Shape Analysis: Models and Algorithms*. Springer-Verlag, 2011.
- [68] Shigeru Owada and Jun Fujiki. DynaFusion: A modeling system for interactive impossible objects. In *International Symposium on Non-Photorealistic Animation and Rendering*, pages 65–68, 2008.

- [69] Yoav I. H. Parish and Pascal Müller. Procedural modeling of cities. In *SIGGRAPH 2001*, pages 301–308.
- [70] Helmut Pottmann, Andreas Asperl, Michael Hofer, and Axel Kilian. *Architectural Geometry*. Bentley Institute Press, 2007. 724 pages, ISBN 978-1-934493-04-5.
- [71] Helmut Pottmann, Yang Liu, Johannes Wallner, Alexander Bobenko, and Wenping Wang. Geometry of multi-layer freeform structures for architecture. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007.
- [72] Helmut Pottmann, Alexander Schiftner, Pengbo Bo, Heinz Schmiedhofer, Wenping Wang, Niccolo Baldassini, and Johannes Wallner. Freeform surfaces from single curved panels. *ACM Transactions on Graphics (SIGGRAPH 2008)*, 27(3), 2008.
- [73] Kenneth Rose, Alla Sheffer, Jamie Wither, Marie-Paule Cani, and Boris Thibert. Developable surfaces from arbitrary sketched boundaries. In *Symposium on Geometry Processing (SGP)*, pages 163–172, 2007.
- [74] R. Sauer. *Differenzengeometrie*. Springer, 1970.
- [75] Guillermo Savransky, Dan Dimerman, and Craig Gotsman. Modeling and rendering Escher-like impossible scenes. *Computer Graphics Forum*, 18(2):173–179, 1999.
- [76] Doris Schattschneider and Michele Emmer, editors. *M.C.Escher’s Legacy: A Centennial Celebration*. Springer, 2003.
- [77] Alexander Schiftner, Mathias Höbinger, Johannes Wallner, and Helmut Pottmann. Packing circles and spheres on surfaces. *ACM Trans. Graph. (SIGGRAPH ASIA 2009)*, 28(5), 2009.
- [78] Idan Shatz, Ayellet Tal, and George Leifman. Paper craft models from meshes. *The Visual Computer: International Journal of Computer Graphics*, 22(9):825–834, 2006.
- [79] Alex Shtof, Alexander Agathos, Yotam Gingold, Ariel Shamir, and Daniel Cohen-Or. Geosemantic snapping for sketch-based modeling. *Computer Graphics Forum (Eurographics 2013)*, 32(2):245–253, 2013.
- [80] Mayank Singh and Scott Schaefer. Triangle surfaces with discrete equivalence classes. *ACM Tran. Graphics (Proc. of SIGGRAPH)*, 29(3), 2010.

REFERENCES

- [81] Peng Song, Chi-Wing Fu, and Daniel Cohen-Or. Recursive interlocking puzzles. *ACM Transactions on Graphics (SIGGRAPH Asia 2012)*, 31(6):128:1–128:10, December 2012.
- [82] Peng Song, Chi-Wing Fu, Prashant Goswami, Jianmin Zheng, Niloy J. Mitra, and Daniel Cohen-Or. Reciprocal frame structures made easy. *ACM Transactions on Graphics (SIGGRAPH 2013)*, 32(4):94:1–94:13, 2013.
- [83] Jos Stam. Aperiodic texture mapping. Tech. Rep. R046, 1997. European Research Consortium for Informatics and Mathematics.
- [84] Markus Steinberger, Michael Kenzel, Bernhard Kainz, Joerg Mueller, Peter Wonka, and Dieter Schmalstieg. On-the-fly generation and rendering of infinite cities on the GPU. *Computer Graphics Forum*, 33(2):105–114, 2014.
- [85] Sachiko Tsuruno. The animation of M.C. Escher’s “Belvedere”. In *ACM SIGGRAPH 97 Visual Proceeding*, page 237, 1997. Presented at Siggraph Electronic Theater 1997.
- [86] Diego Uribe. A set of impossible tiles, 2001. The third international conference Mathematics and Design, Available at <http://im-possible.info/english/articles/tiles/tiles.html>.
- [87] ustwo studio Ltd. Monument valley, 2014. <http://www.monumentvalleygame.com/>.
- [88] Valve Corporation. Portal and Portal 2, 2007. <http://www.valvesoftware.com/games/portal.html> and <http://www.valvesoftware.com/games/portal2.html>.
- [89] Jarke J. van Wijk. Symmetric tiling of closed surfaces: visualization of regular maps. *ACM Trans. Graph. (SIGGRAPH 2009)*, 28(3), 2009.
- [90] Goo-Shun Wang. Hallucii, 2006. <http://www.imdb.com/title/tt1139791/>.
- [91] Hao Wang. Proving theorems by pattern recognitionii. *Bell System Technical Journal*, 40(1):1–41, 1961.
- [92] Wenping Wang, Yang Liu, Dongming Yan, Bin Chan, Ruotian Ling, and Feng Sun. Hexagonal meshes with planar faces. Technical report, 2008. HKU CS Tech Report TR-2008-13.

- [93] Tim Weyrich, Jia Deng, Connelly Barnes, Szymon Rusinkiewicz, and Adam Finkelstein. Digital bas-relief from 3D scenes. *ACM Transactions on Graphics (SIGGRAPH 2007)*, 26(3), 2007. Article 32.
- [94] H. Wolfson, E. Schonberg, A. Kalvin, and Y. Lamdan. Solving jigsaw puzzles by computer. *Annals of Operations Research*, 12:51–64, 1988.
- [95] Peter Wonka, Michael Wimmer, François Sillion, and William Ribarsky. Instant architecture. *ACM Transactions on Graphics (SIGGRAPH)*, 22(3):669–677, 2003.
- [96] Tai-Pang Wu, Chi-Wing Fu, Sai-Kit Yeung, Jiaya Jia, and Chi-Keung Tang. Modeling and rendering of impossible figures. *ACM Transactions on Graphics*, 29(2):13:1–15, 2010.
- [97] Kai Xu, Hao Zhang, Andrea Tagliasacchi, Ligang Liu, Guo Li, Min Meng, and Yue-shan Xiong. Partial intrinsic reflectional symmetry of 3D shapes. *ACM Trans. Graph. (SIGGRAPH ASIA 2009)*, 28(5), 2009.
- [98] Dong-Ming Yan, Yang Liu, and Wenping Wang. Quadric surface extraction by variational shape approximation. In *Geometric Modeling and Processing*, pages 73–86. Springer Berlin / Heidelberg, 2006.
- [99] Miaojun Yao, Zhili Chen, Linjie Luo, Rui Wang, and Huamin Wang. Level-set-based partitioning and packing optimization of a printable model. *ACM Transactions on Graphics (SIGGRAPH Asia 2015)*, 34(6):214:1–214:11, 2015.
- [100] Jane Yen and Carlo Séquin. Escher sphere construction kit. In *Proceedings of the 2001 Symposium on Interactive 3D Graphics*, I3D '01, pages 95–98. ACM, 2001.
- [101] Ruo Zhang, Ping-Sing Tsai, James Edwin Cryer, and Mubarak Shah. Shape from shading: A survey. *Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, 21(8):690–706, 1999.
- [102] Henrik Zimmer, Marcel Campen, David Bommes, and Leif Kobbelt. Rationalization of triangle-based point-folding structures. *Computer Graphics Forum*, 31(2pt3):611–620, 2012.

Publications

Published:

- **Chi-Fu Lai**, Sai-Kit Yeung, Xiaoqi Yan, Chi-Wing Fu*, and Chi-Keung Tang. 3-D Navigation on Impossible Figures via Dynamically Reconfigurable Maze. IEEE Transactions on Visualization and Computer Graphics, Vol. 22, No. 10, pp. 2275-2288, 2016.
- Xiaoqi Yan, **Chi-Fu Lai**, and Chi-Wing Fu*. Visual Signature of High-Dimensional Geometry in Parallel Coordinates. IEEE Pacific Visualization Symposium (PacificVis), 65-72, 2014.
- Shi-Qing Xin, **Chi-Fu Lai**, Chi-Wing Fu*, Tien-Tsin Wong, Ying He, and Daniel Cohen-Or. Making Burr Puzzles from 3D Models. ACM Transactions on Graphics (SIGGRAPH 2011), Article No. 97, Vol. 30, No. 4, 2011.
- Chi-Wing Fu*, **Chi-Fu Lai**, Ying He, and Daniel Cohen-Or. K-set Tiling Surfaces. ACM Transactions on Graphics (SIGGRAPH 2010), Article no. 44, Vol. 29, no. 3, 2010.

*Ph.D. advisor