3D Navigation on Impossible Figures via Dynamically Reconfigurable Maze

Chi-Fu William Lai, Sai-Kit Yeung, *Member, IEEE*, Xiaoqi Yan,
Chi-Wing Fu, *Member, IEEE*, and Chi-Keung Tang, *Senior Member, IEEE*

Abstract—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. In this paper, we present a new approach towards 3D gaming with impossible figures, delivering for the first time navigation in 3D mazes constructed from impossible figures. Such result cannot be achieved by previous research work in modeling impossible figures. To deliver seamless gaming navigation and interaction, we propose i) a set of guiding principles for bringing out subtle perceptions and ii) 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.

Index Terms—Computational geometry and object modeling, impossible figure, game, dynamic and procedural modeling

1 Introduction

The impossible figures designed by Escher [1] (see Fig. 1a for the Infinite Staircase) have intrigued generations of scholars, artists, game developers, and entertainers. One popular example was the box office hit *Inception* [2] in 2010, where a set of staircases were carefully arranged (see Fig. 1b) 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 Figs. 2a and 2c; 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 Fig. 1b).

Impossible figures have been used as mazes in games, e.g., *Echochrome* [3]. However, all existing games, including the popular action RPG game *Diablo II* [4] and the recent award-winning game *Monument Valley* [5] (see Fig. 1c) 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 [6] formulated a constraint solver using multiple meshes to obtain a rotatable 2D impossible figure, while Wu et al. [7] employed thin plate spline to model an impossible figure via constrained deformation. However, none of them considered 3D navigation over a virtual world built from an impossible figure.

1.1 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

- *C.-F.W. Lai is with the School of Computer Engineering, Nanyang Technological University, Singapore, and Singapore University of Technology and Design. E-mail: la0001fu@ntu.edu.sg.*
- *S.-K. Yeung is with the Singapore University of Technology and Design, Singapore. E-mail: saikit@sutd.edu.sg.*
- *X. Yan is with the School of Computer Engineering, Nanyang Technological University, Singapore. E-mail: yanx0005@ntu.edu.sg.*
- *C.-W. Fu is with the Department of Computer Science and Engineering, Chinese University of Hong Kong, and the School of Computer Engineering, Nanyang Technological University, Singapore. E-mail: cwfu@acm.org.*
- *C.-K. Tang is with the Department of Computer Science and Engineering, Hong Kong University of Science and Technology, Hong Kong. E-mail: cktang@cse.ust.hk.*

Manuscript received 8 Dec. 2014; revised 19 Nov. 2015; accepted 30 Nov. 2015. Date of publication 0 . 0000; date of current version 0 . 0000. Recommended for acceptance by S. Takahashi.
For information on obtaining reprints of this article, please send e-mail to: reprints@ieee.org, and reference the Digital Object Identifier below.
Digital Object Identifier no. 10.1109/TVCG.2015.2507584

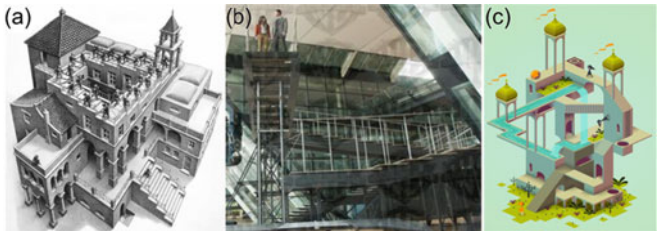


Fig. 1. (a) Ascending & Descending, M. C. Escher's painting showing monks walking indefinitely in the INFINITE STAIRCASE; (b) a making-of photograph showing the INFINITE STAIRCASE in movie *Inception*; and (c) a game level in *Monument Valley*, embedding an impossible-figure structure.

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 [6] or Wu et al. [7], since we cannot twist the geometry of an impossible figure nor break it at fixed gaps, see Fig. 2. 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.

1.2 Our Approach

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

1) We revisit the notion of impossible figures and present a set of *guiding principles* for producing seamless 3D navigation and interaction experience when building a gaming maze from an impossible figure (Section 3).
2) We develop *dynamically-reconfigurable maze*, which has the following three novel parts:
   a) The first part is an automatic method that analyzes and turns line-art images of impossible figures into box-based 3D structures (Section 4).
   b) Then, we propose a novel procedure that dynamically reconnects and re-structures an impossible-figure maze based on the user's view, so that we can achieve the guiding principles and enable seamless single-player navigation and interaction (Section 5).
   c) Lastly, we adopt and extend the basic modeling technique for different game maze types (Section 6): 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.

2 RELATED WORK

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 [8]. Later,

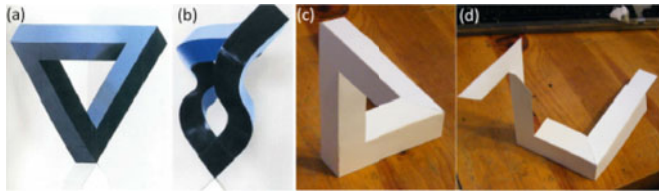


Fig. 2. (a) PENROSE TRIANGLE photographed at a specific view of (b) a highly-twisted solid; (c) another PENROSE TRIANGLE based on (d) a disconnected paper model.

Escher [1] popularized impossible figures and embedded them in a number of his famous drawings, e.g., see Fig. 1a.

In general, there are four classes of impossible figures, each with a different construction mechanism for achieving the perceptual impossibility: *depth contradiction*, *depth interposition*, *disappearing normals*, and *disappearing space*, see Fig. 3, Ernst [9], and Wu et al. [7]. In this work, we cannot deal with the last class since it does not have a clear boundary between the foreground and background, so offering no maze for one to walk over. Below, we discuss various areas of work related to impossible figures and 3D dynamic and procedural modeling.

2.1 Computer Games with Impossible Figures

Alexeev's impossible world website [10] 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* [11] 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* [4] embedded an impossible figure as a part of a huge 2.5D gaming terrain in a level called *Secret Sanctuary*. Though both games used impossible figures in some interesting ways, they show only a fixed view of an impossible figure with an axonometric projection without offering 3D camera controls like most conventional 3D games. A recent game on mobile platform called *Monument Valley* [5] (see Fig. 1c) allows us to rotate and play with a 3D object built by breaking an impossible figure at fixed gaps. This is similar to the paper model shown in Figs. 2c and 2D. 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 sharp contrast, this work, for the first time, constructs and provides a 3D gaming maze from an impossible figure. By our approach, the user can walk on an impossible-figure maze with first/third-person 3D perspective viewing and camera control during the immersive navigation.

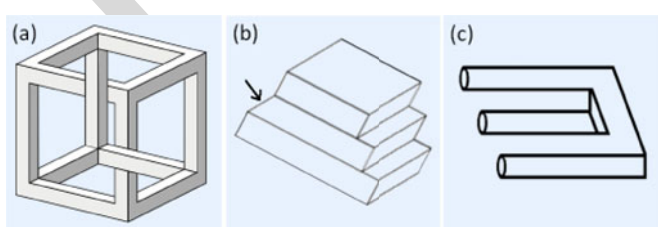


Fig. 3. Classes of impossible figures: i) *depth interposition*, e.g., IMPOSSIBLE CUBOID (a); ii) *disappearing normals*, e.g., IMPOSSIBLE STAIRCASE (b); iii) *disappearing space*, e.g., IMPOSSIBLE TRIDENT (c); and iv) *depth contradiction*, e.g., PENROSE TRIANGLE (Fig. 2a).

2.2 Modeling and Rendering Impossible Figures

Previous work in computer graphics research focused mainly on generating plausible views of an impossible figure. Tsuruno [12] created an interesting computer animation that shows different views of *Belvedere*, one of Escher's drawings [1] with impossible figures. Later, Savransky et al. [13] proposed using relative transformations from user input to model impossible figures.

Khoh and Kovesi [14] 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 [6] developed a more general optimization method to broaden the range of plausible views, and built a game-like system called theRelativity [15], [16]. However, the range of viewing angles on impossible figures is still limited. Hence, Wu et al. [7] 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 [17] 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. [7] and Elber [17] are highly twisted in the world space, so we cannot apply these methods to model a navigable virtual world as in our case. Compared to the above works, we present a novel approach that can offer *3D immersive gaming adventure with impossible figures*. Such result cannot be achieved by any of the above computer graphics methods for modeling impossible figures.

2.3 3D Dynamic and Procedural Modeling

Parish and Müller [18] proposed *CityEngine*, a L-system-based procedural modeling engine capable of generating a large-scale virtual city. After that, Wonka et al. [19] developed a grammar-based method to automatically generate complex buildings, while Müller et al. [20] improved it with a shape grammar, especially considering façade details. Later, Lipp et al. [21] proposed an interactive visual editing paradigm for shape grammars.

More recently, Danihelka et al. [22] developed a real-time procedural modeling method that can support multiple users navigation with consistent views among the users. Steinberger et al. [23] 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.

3 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.

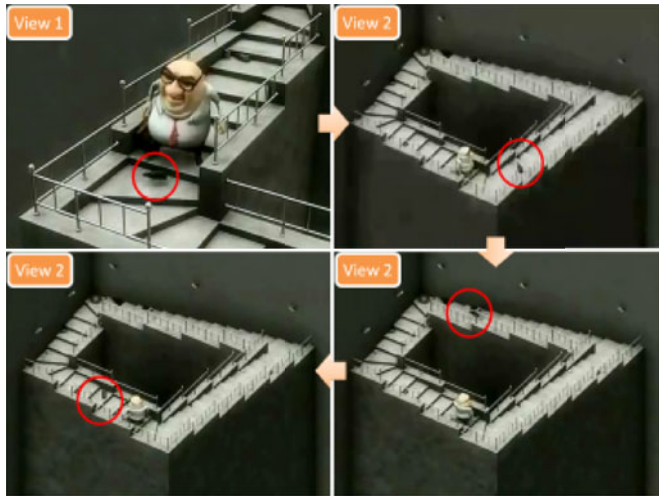


Fig. 4. A key frame sequence (following the orange arrows) in 3D animation *Hallucii*, showing a drunk man who walks over a 3D infinite staircase. Top-left image shows a local view on the drunk man while the others show the entire staircase.

3.1 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.

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. So, 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 contradiction, or subtle structural inconsistency, 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 Fig. 3.

3.2 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* [24], a conventional 3D animation, as an example, see Fig. 4. 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 Fig. 4(top-left)). During the walk, the drunk man kicks a glass bottle downstairs (see top-left image

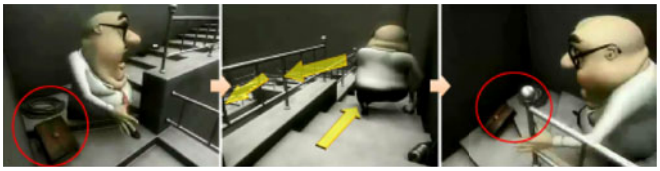


Fig. 5. Another key frame sequence from *Hallucii*, showing that the drunk man finds the *same* briefcase again at the *same* corner after completing one cycle down the 3D infinite staircase. The yellow arrows indicate his walking direction.

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 Fig. 5). 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.

3.3 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 Fig. 4(top-left) for a local view and Fig. 4(top-right) 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 Figs. 4(top-left) and 5. In other words, we should not see any gap or convolved surface like Figs. 2b and 2d.
- *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* (Fig. 5) 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).

4 Geometry Representation

4.1 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 Fig. 6a. 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.[1]

i) Geometry representations. Given the impossible-figure input, our next step is to convert it into a geometry representation, see Fig. 6b. Since different kinds of geometry representation could be needed for different working scenarios, we support two kinds of geometry representations:

- *Image-based triangles* (Fig. 6b(top)): Here we construct a gradient-aware triangulation (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 Fig. 6b.
- *Solid Primitives* (Fig. 6b (bottom)): Other than image-based triangles, which are surface-based open meshes, we develop an automatic method (see Section 4.2) 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 Fig. 6b.

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 5. 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 create 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. In addition, 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.

4.2 Constructing Box-based Solid 3D Primitives

We develop an automatic method to arrange and tile connected 3D boxes over an impossible-figure image, see the

1. To prepare a normal map image, we develop a simple program that renders a cube in orthogonal view with faces colored based on their normal vectors in the screen space (see Figs. 7a and 7b). 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 Fig. 7c).

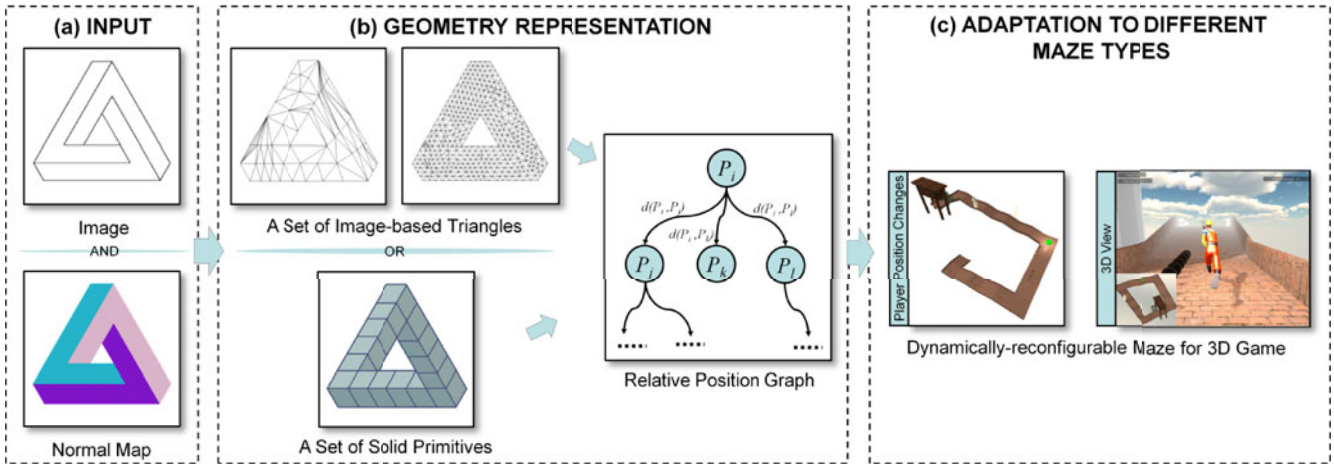


Fig. 6. Our computational pipeline (a) takes an impossible figure (RGB image and normal map) as input, (b) converts it into a set of triangles or primitives (later encoded by a relative position graph), and then (c) adapts it for different maze types.

result in Fig. 6b(bottom). Here we assume that the input impossible figure is an orthogonal projection with three major axis directions in image space, see the RGB arrows in Fig. 8a. 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 Procedure 1:

Procedure 1. PARTITIONING THE IMPOSSIBLE FIGURE

```
Identify all contour lines in the impossible-figure image
Identify three major axis directions among contour lines
Assign a direction to each contour line
for each contour line do
    for each endpoint p of the line do
        while p not extend out of the figure foreground and
        not at a Y-junction /* See Fig. 8(b) */ do
            /* Case 1: Occluded Linkage (See Fig. 9a) */
            if p is at a T-junction then
                if a contour line is found in p's extend direction
                then
                    join the two lines
                    update p as far endpoint of line found
                else
                    break /* end while loop */
                end if
            end if
            /* Case 2: Join lines (See Figs. 8b & 8c)) */
            if p touches a parallel contour line then
                join the two lines
                update p as far endpoint of line found
                continue /* goto step 6 */
            end if
            /* Case 3: Change direction (See Fig. 8d) */
            if p is about to extend to a perpendicular face then
                modify p's extend direction
            end if
            Extend p until it intersects/touches a contour line
        end while
    end for
end for
```

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. 8b). By further analyzing the line orientations, we can find the three major axis directions in the orthogonal projection (Fig. 8a) and assign a direction to each contour line.

Lines 4-29: Extend Contour Lines. Second, 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 Fig. 8b) or reach the foreground boundary:

1) *Lines 7-15: Occluded (Layered) Linkage.* For geometric structure 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 Fig. 9a: 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.
2) *Lines 16-21: Join parallel lines.* When extending a contour line, if its endpoint meets another contour line, we should join the two lines and continue the line extension from the other end of the joined line, see Figs. 8b and 8c, so we can produce a larger quadrangle.

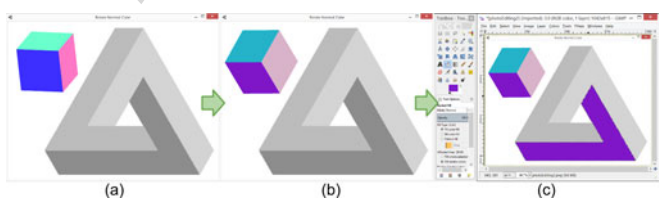


Fig. 7. (a and b): our simple rotating-cube program for finding normal map colors of impossible figure images. (c) we then use GIMP to fill corresponding regions in the figure image.

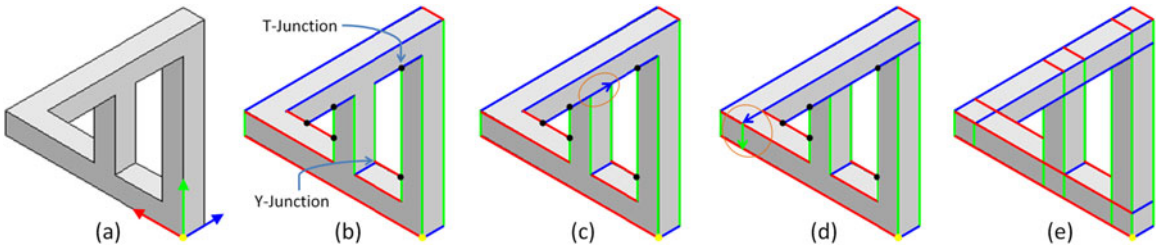


Fig. 8. Partition the foreground region of the input impossible figure into quadrangles: (a) an input impossible figure, and the three major axes in orthogonal projection; (b) extract contour lines and identify their directions; (c-e) extend the contour lines one by one in the image space to partition the figure, see (e) for the result and Procedure 1 for the pseudo code outline.

3) *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 make sense if we continue with the same line direction, see Fig. 8d. Rather, we should adjust the direction along which the line 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 Fig. 8e.

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 showing 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 Fig. 10a, and *partial quadrangles*, see the marked quadrangles in Fig. 10b. Note that full quadrangles are simply fully visible faces of boxes in the impossible-figure structure.

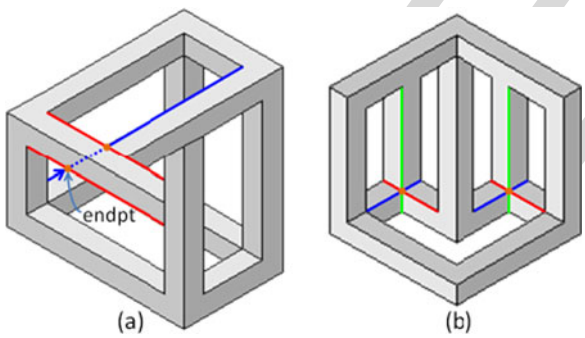


Fig. 9. (a) Occluded linkage: the dotted line between the two visible blue lines; (b) Star-junction (orange dots), where six contour lines meet and a cube is hidden behind.

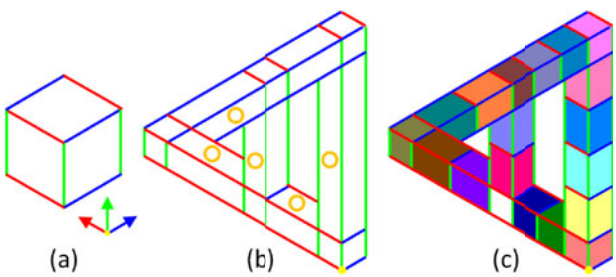


Fig. 10. (a) A single box. (b) orange circles mark partially-occluded faces (partial quadrangles) of boxes. (c) box-fitting results: we assign the same color to faces of the same box.

- Second, there are four kinds of boxes: boxes with three, two, one, and zero "fully visible" faces (full quadrangles), see Fig. 11(top row), (middle row), (bottom row), and the star junctions in Fig. 9b, respectively.

i) Initialization. The box-fitting stage starts by searching for a Y-junction shared by three full quadrangles, see Fig. 11(top). If found, we create a *cornerstone box* with such a quadrangle. Otherwise, we look for a 2-face pattern, see Fig. 11 (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 Fig. 11 to reconstruct a neighboring box. Note that in the relation 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 Fig. 9b, since the hidden box behind a star junction cannot be recovered by the patterns shown in Fig. 11. Hence, whenever we reach a star junction, we reconstruct the hidden box by using the three shorter edges around it.

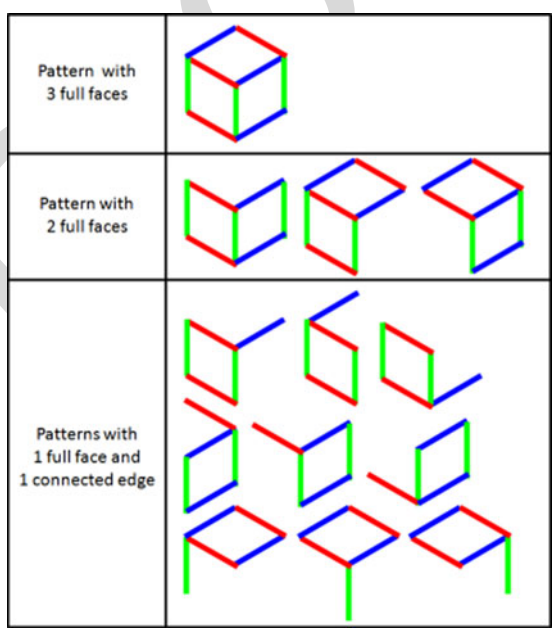


Fig. 11. Three different groups of patterns for box-fitting; note that these patterns can be rotated during the matching.

iv) Subdivision. Lastly, after the traversal is complete, we avoid long boxes by subdividing them into smaller boxes, see Fig. 10c for the result. This is to give greater flexibility to the dynamic modeling method to be presented in Section 5.

5 Dynamically-Reconfigurable Maze

Using the relative position graph as a representation of the input impossible figure, we can dynamically reconfigure and construct a maze geometry of the impossible figure, which is a *view-dependent geometry*, subject to user's *viewpoint*.

Procedure 2 presents the pseudo code of our modeling method given P_c, see also Table 1 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 user's avatar is located (see Sections 6.2.1, 6.2.2 and 6.2.3). 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:

Procedure 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: $p(P_c) = \emptyset$ /* P_c is root */
4: position P_c as it is in 3D
5: **while** $Q \neq \emptyset$ **do**
6: /* step 1: connect P_f to parent of P_f (if any) */
7: P_f = pop front of Q
8: mark P_f visited
9: **if** $p(P_f) \neq \emptyset$ **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 $v \in$ vertices of P_f **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 $P_i \in$ **Nb**(P_f) **do**
22: **if** P_i not visited **then**
23: **if** the case of image-based triangles **then**
24: $tempD = d(P_f) + d(P_f, P_i)$
25: **else**
26: $tempD = d(P_f) + \| \vec{d}(P_f, P_i) \|$
27: **end if**
28: **if** $d(P_i) > tempD$ **then**
29: $d(P_i) = tempD$
30: $p(P_i) = P_f$
31: if $P_i \notin Q$, add 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**

TABLE 1
Notations and Functions Used in Procedure 2

Notations/Functions:	Meaning:
P_c	the specific primitive containing the point of interaction, e.g., the current user 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
Nb(P_i)	set of neighboring nodes of P_i in the graph
sortQueue(Q)	a procedure to sort the primitives in Q in ascending order of $d(P_i)$
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
connectTriangle (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 Fig. 6).
For the case of solid primitives:	
$\vec{c}(P_i)$	P_i's centroid in 3D space relative to P_c

5.1 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 Figs. 2c and 2d, 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 Figs. 2a and 2b) 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 Procedure 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

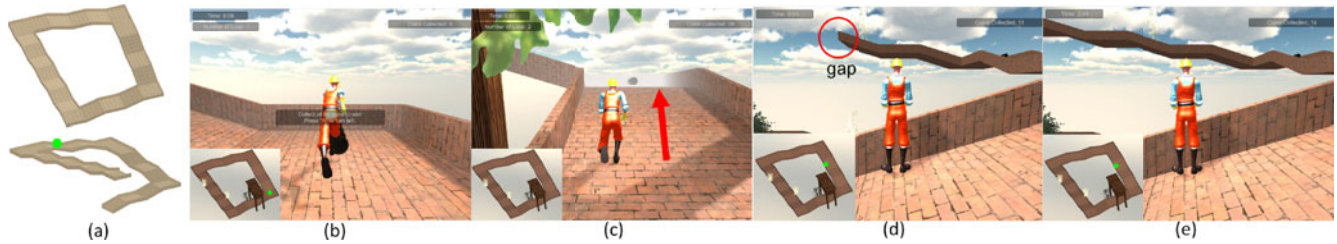


Fig. 12. Endless runner game prototype: (a) the input impossible figure (top) and the maze model dynamically reconstructed by Procedure 2 w.r.t. P_c (green dot); (b & c) the starting and ending views, respectively, where the game avatar keeps running upwards endlessly to collect the coins on the impossible-figure maze; (d) without the adaptation strategy, the gap can be seen when the user looks around; and (e) the adaptation strategy can help avoid the gap and achieve *seamless viewing*.

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 in the relative position graph, see Fig. 6b, after completing Procedure 2, these unconnected edges should be far away from P_c, thus keeping the gaps far from the avatar. Furthermore, when the avatar moves, we update P_c, and apply Procedure 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.

5.1.1 Numerical Issues in Connections

In step 1 of Procedure 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.

5.1.2 Performance

Procedure 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 3.20 GHz and 9 GB 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.

6 ADAPTATION TO DIFFERENT MAZE TYPES

The last part of our approach adapts and extends the basic modeling procedure presented in Section 5 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 Procedure 2 to dynamically generate the maze (broken) geometry subject to the avatar, e.g., the red box in Fig. 13b, 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 Fig. 12d), thus violating the *seamless viewing* principle (see Section 3). 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 Fig. 6b) except Figs. 12 and 15, which are surface-based models mimicking the mazes in Hallucii [24] and a specific scene in Diablo II, respectively.

6.1 Achieving Seamless Viewing

Our key idea to achieve *seamless viewing* is by modifying the broken geometry from Procedure 2, i.e., by pushing the gaps out of user's view frustum. However, we still keep the broken geometry from Procedure 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. 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* (Fig. 12e) upon view changes.

6.2 Different Generic Maze Types

6.2.1 Type 1: A Maze of Corridors

Since we can take solid primitives as inputs, we can produce a maze of corridors, where the users 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 Fig. 13a. By our method, the user can never reach and see gaps in the scene in the navigation (Figs. 13c, 13d, 13e) even though the scene is actually broken somewhere (see Fig. 13b). 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 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 [25], and Antichamber [26], 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

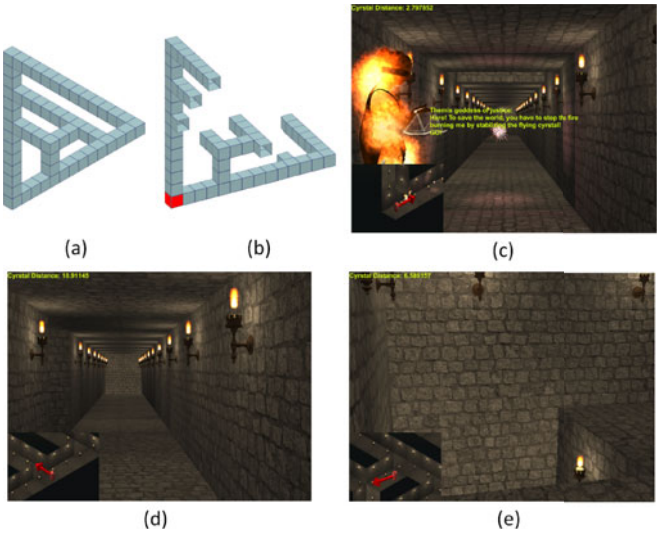


Fig. 13. Chasing game prototype: (a) input impossible figure (box-based solid primitives); (b) the maze model built from Procedure 2 w.r.t. P_c (red box); and (c-e) subsequent screenshots, demonstrating *subtle impossibility perception* when we move over the maze, see also the mini-map (lower-left).

in a physically-impossible manner; though this idea is innovative, there is no dynamic scene reconstruction and the scene is essentially a possible object attached with pairs of teleport holes.

6.2.2 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 user is too large, it may be possible for the user 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, featuring fast movement through an impossible-figure world, see Fig. 12 for the screenshots and the impossible figure we employed. By using Procedure 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 user can look around and gaps can be avoided accordingly in user's view (by shifting gaps out of the view frustum), see Figs. 12d and 12e. Please see our supplementary video, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TVCG.2015.2507584, for the navigation experience.

Since the infinite staircase is repetitive by nature, an alternative approach to model it is by duplicating multiple instances of its structure and vertically connecting them into a mega staircase structure. Fundamentally, this duplication approach shares the same spirit as our approach in terms of reconnecting scene components for achieving seamless navigation and viewing. However, it connects multiple instances of the entire maze (per-maze), while we connect boxes in the maze in a finer scale. Hence, we can see the following issues with this alternative approach: i) structure duplication requires us to explicitly define a possible solid by defining gaps to break the impossible figure structure into a possible solid, so that the rendering system can connect instances of the maze at the gaps; and ii) we need to render multiple instances of the entire maze if the avatar stands on one instance and sees another instance(s) in the view. Hence, our box-based approach can offer higher flexibility in terms of modeling the impossible-figure structure, allowing us to intuitively represent the impossible-figure structure as a graph and reconnect boxes in the graph according to user's view.

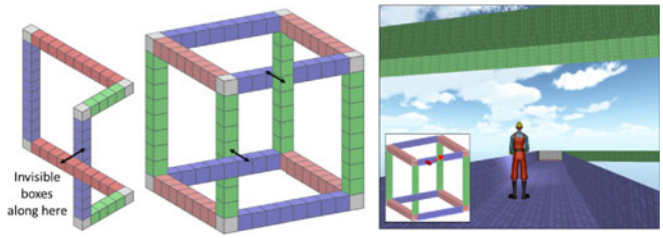


Fig. 14. Left: our box fitting method can also work with impossible figures of the depth interposition class. Right: by manually connecting the interposition links with invisible boxes (see the double-ended arrows on the left), our method can also work with this class of impossible figure.

Note that our method can also support mazes built with the "depth interposition" class (see Fig. 14). 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 Fig. 14(left)). Hence, our dynamic modeling method will force an unnatural positioning between the scene geometries around the interposition links.

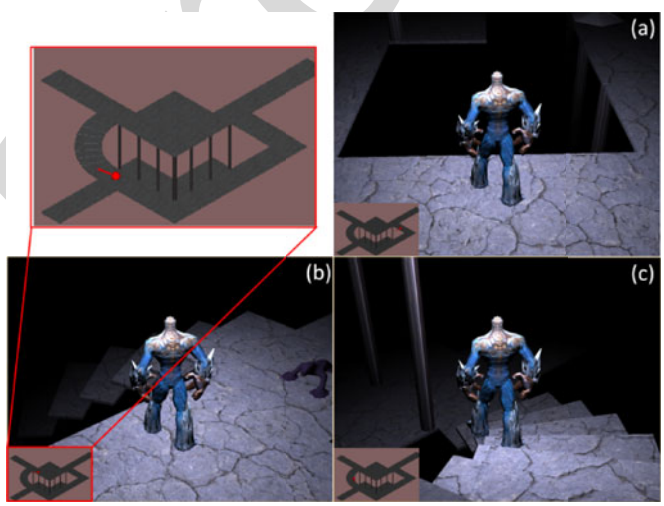


Fig. 15. Screenshots from the game prototype (Role-Playing Game) that mimics the Arcane Sanctuary level in Diablo II (which is simply a 2.5D impossible world). Top-left: an example position and view direction of the avatar. (a-c): our method can reconstruct this impossible-figure world in 3D.

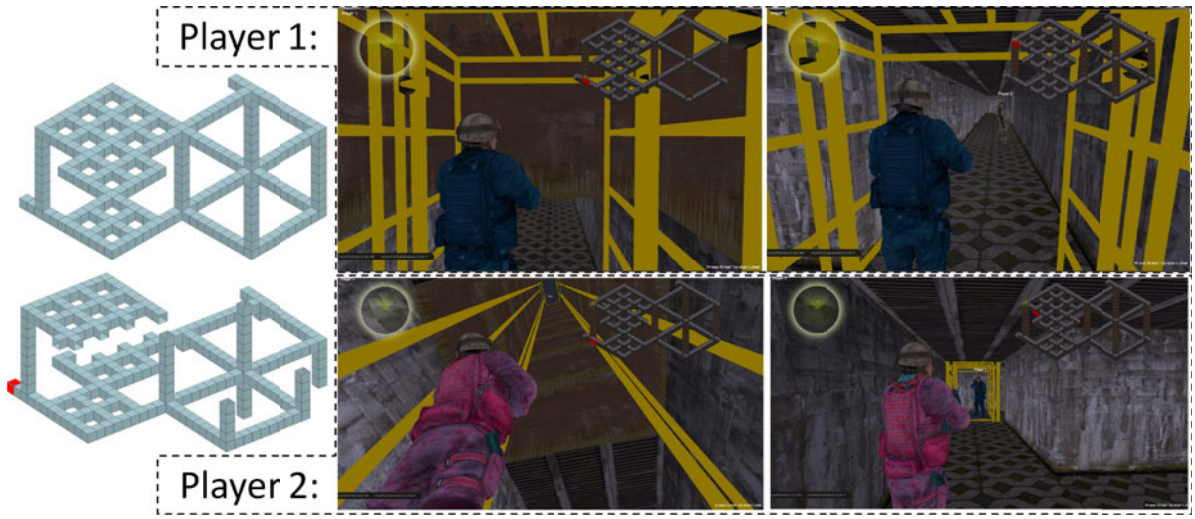


Fig. 16. Multi-player shooting game prototype: The nontrivial impossible figure (top-left) from which the maze of corridors is built; and the maze model (bottom-left) reconfigured by Procedure 2 w.r.t. P_c (the red box); Player 1 (in blue) took the lift to move up to a seemingly "upper" level (Player 1: top-left). Next, Player 2 (in brown) entered the game and found Player 1 inside the lift above him (Player 2: bottom-left). 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 the other (Players 1 and 2: right column).

Game Prototype ii): Role-Playing Game. The other open-space prototype we built is a role-playing game environment that mimics the 2.5D Arcane Sanctuary level in *Diablo II* [4]. Here, we use a third person view and the user can freely look around in the 3D virtual world, see Figs. 15a, 15b, and 15c for screenshots with corresponding mini-maps, which indicate the avatar's corresponding location and viewing direction.

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 Fig. 15a, 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.

6.2.3 Type 3: Multiple Platforms

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 Procedure 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 Fig. 16. 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 Fig. 16.

Achieving multi-player interaction. We propose an egocentric approach to resolve the above issue: i) we first apply Procedure 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 Procedure 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. Fig. 16 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, available online), 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

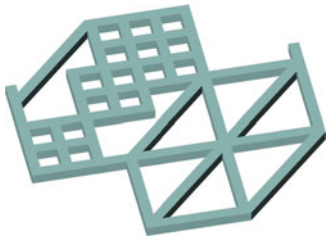


Fig. 17. A flattened figure resembling the impossible figure input in the multi-player shooting game prototype.

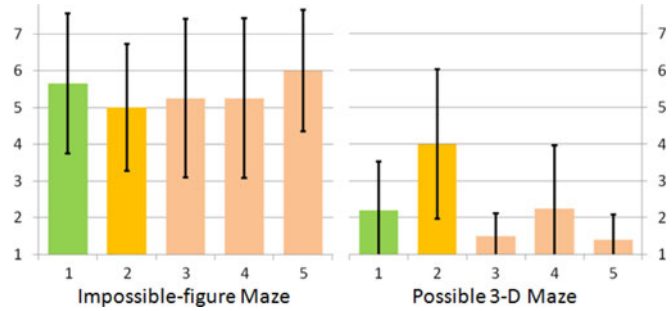


Fig. 18. Statistical results gathered from the 20 participants who tried our multi-player gaming prototype: mean and standard deviation of variables (1)-(5) for the case of impossible-figure maze (left) and possible 3D maze (right).

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.

7 Discussion

7.1 User Study

We conducted a user study to examine user's cognition and experience when navigating in a virtual 3D world built from an impossible figure.

Preparation. First, we slightly modified the multi-player shooting game prototype presented in Section 6.2.3 with the followings: i) instead of showing the whole maze, the mini-map on the gaming interface shows only a local portion of the maze surrounding the gaming avatar; and ii) we created a possible 3D maze (see Fig. 17) that resembles the impossible-figure maze used in the multi-player shooting game prototype. This maze was 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 seven 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 presented to the group the knowledge of impossible figures using the contents in Section 3: "The Notion of Impossible Figure." We then showed the Infinite Staircase in Fig. 4 (top-right) to the participants: "*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 showed to them the impossible figure for building the maze in the multi-player shooting game, i.e., Fig. 16(top-left), and discussed with them to ensure that they understood the perceptual impossibility in the figure. The two participants in a group were then randomly assigned as Player A and Player B, sat on opposite sides of a table, and tried our multi-player shooting game prototype on his/her own computer screen.

Practice session. After the tutorial session, the two participants were given 3 minutes to individually try the game in a single player mode, so that they can get familiar with the gaming controls. Moreover, we prevented them from understanding the maze structure by restricting them to explore only a very small local area around the starting location in the maze.

Gaming sessions. There were two gaming sessions for each group of participants: one with the impossible-figure maze in Fig. 16 and the other with the possible 3D maze in Fig. 17. The order of these two gaming sessions was 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 did the other way around. Note that the participants did not know the actual order.

Tasks. After the practice session, the first gaming session started by randomly assigning a starting location to the avatar of each participant, while keeping the two random locations to be far away from each other in the maze. Then, the two participants were given the following tasks:

1) Explore the virtual world.
2) Find and shoot the other player (until no more hit point).

After one of the players finished the tasks, we ended 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 asked each participant to fill a questionnaire to examine their experience in the session. The questionnaire consists of five variables designed based on [27] using a Likert scale of 1 to 7 (very unlikely 1-very likely 7):

Perceived Impossibility (1-7):

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

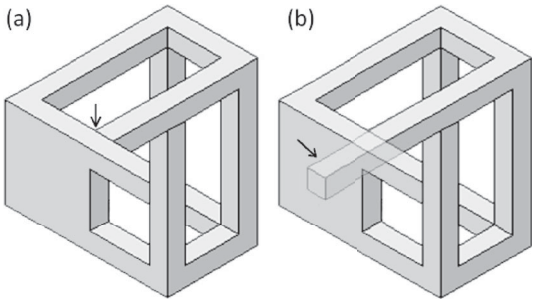


Fig. 19. (a) An input impossible figure with a covered region behind the wall (see arrow); there could be many possible ways of reconstruction; (b) extra user input is needed, if we want to reconstruct the hidden structure marked by the arrow; otherwise, our current method will leave it unconnected.

In the questionnaire, the five variables attempt to explore the navigation experience of the participants: variable (1) directly asks the participants on whether they find the maze to be an impossible figure or not; variable (2) examines if the participants are aware of the mental process of reconstructing the virtual world, while variables (3-5) 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.

Results. Fig. 18 presents the statistical results, showing the mean and standard deviation of variables (1)-(5). First of all, from the plots of variable (2) 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 (3)-(5) 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.

Lastly, we performed a paired t-test on variable (1) 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 H0 is *the mean values of variable (1) 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 percent. Hence, we can reject H0 at the significance level and suggest that there is a statistical difference between participants' perception on the two mazes.

7.2 Limitations

7.2.1 Input Impossible Figures

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 Fig. 3. 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 Fig. 3b, 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 Fig. 3c, so our method cannot form a consistent geometry representation for impossible figures of this class neither.

7.2.2 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 line (Procedure 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 Fig. 19a: in this case, our current method cannot automatically reconstruct and connect the hidden structure marked by the arrow in Fig. 19b. In this situation, extra user input is needed.

7.2.3 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 INFINITE STAIRCASE), 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.

7.3 Other Applications

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

8 CONCLUSION

This paper presents novel computational methods that construct virtual mazes using impossible figures for 3D gaming. Our method delivers seamless interaction and viewing with impossible figures, so that we 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 develop an automatic method to analyze 2D impossible-figure images and construct connected 3D boxes to model the figure for supporting 3D gaming. Third, we devise a real-time procedure to dynamically reconfigure and reconnect the maze model subject to the user's position, thereby achieving seamless navigation over the maze even though a complete maze cannot be modeled plausibly in 3D

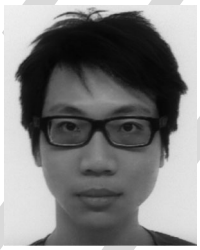
(without gap and twist). Lastly, we adopt and extend the method to support various generic maze types, and develop several game prototypes to demonstrate the applicability of our method.

Acknowledgments

The authors thank anonymous reviewers for the various constructive comments, ustwo studio Ltd. and Goo-Shun Wang for the permission to use the images in Fig. 1c and Figs. 4 and 5, respectively. Figs. 1, 2, 3, 4, and 5 belong to their respective owners. Note that by Copyright Disclaimer Under Section 107 of the Copyright Act 1976, allowance is made for fair use which explicitly applies to use of copyrighted work for research purposes. This work was supported in part by the MOE Tier-2 fund (MOE2011-T2-2-041(ARC 5/12)). Sai-Kit Yeung is supported by SUTD-ZJU Collaboration Research Grant 2012 (SUTDZJU/RES/03/2012), SUTD-MIT International Design Center Grant (IDG31300106) and MOE Tier-2 fund (MOE2013-T2-1-159). Part of the work was done when Chi-Fu William Lai was visiting SUTD and supported by the National Research Foundation (NRF) Singapore under its Interactive Digital Media (IDM) Strategic Research Programme. They acknowledge the support of the SUTD DManD Centre which is supported by the Singapore NRF.

References

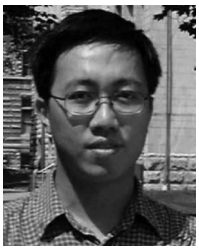
[1] M. C. Escher Foundation. (2005). The official M.C. Escher website [Online]. Available: http://www.mcescher.com
[2] C. Nolan. (2010). Inception [Online]. Available: http://www.imdb.com/title/tt1375666/
[3] Game Yaruoze, SCE Studios Japan, Sony Computer Entertainment. (2008). Echochrome [Online]. Available: http://www.jp.playstation.com/scej/title/mugen/.
[4] Blizzard Entertainment, Inc. (2000). Diablo II [Online]. Available: http://www.blizzard.com/diablo2/
[5] ustwo studio Ltd. (2014). Monument valley [Online]. Available: http://www.monumentvalleygame.com/
[6] S. Owada and J. Fujiki, "DynaFusion: A modeling system for interactive impossible objects," in *Proc. Int. Symp. Non-Photorealistic Animation Rendering*, 2008, pp. 65–68.
[7] T.-P. Wu, C.-W. Fu, S.-K. Yeung, J. Jia, and C.-K. Tang, "Modeling and rendering of impossible figures," *ACM Trans. Graph.*, vol. 29, no. 2, pp. 13:1–13:15, 2010.
[8] Lilly Library. (2014). Swedish designer Oscar Reutersvard's impossible images [Online]. Available: http://libraries.iub.edu/swedish-designer-oscar-reutersv%C3%A4rds-impossible-images
[9] B. Ernst, *Adventures with Impossible Figures*. Stradbroke, England: Tarquin, 1987.
[10] V. Alexeev. (2001). Impossible world - impossible figures in computer games [Online]. Available: http://im-possible.info/english/art/games/index.html
[11] Louis (t). (2012). Adynatopia [Online]. Available: http://www.kongregate.com/games/iiaopsw/adynatopia
[12] S. Tsuruno, "The animation of M.C. Escher's 'Belvedere'," in *Proc. ACM SIGGRAPH Vis.*, 1997, p. 237.
[13] G. Savransky, D. Dimerman, and C. Gotsman, "Modeling and rendering Escher-like impossible scenes," *Comput. Graph. Forum*, vol. 18, no. 2, pp. 173–179, 1999.
[14] C. W. Khoh and P. Kovesi. (1999). Animating impossible objects [Online]. Available: www.csse.uwa.edu.au/∼pk/Impossible/impossible.html
[15] J. Fujiki and S. Owada. (2008). theRelativity. *Proc. ACM SIGGRAPH ASIA Art Gallery: Emerging Technol.*, p. 15, demo movie [Online]. Available: http://jun-fujiki.com/theRelativity/movie/theRelativity.mpg.
[16] J. Fujiki and S. Owada. (2008). theRelativity+inPossible [Online]. Available: http://jun-fujiki.com/theRelativity/index.html
[17] G. Elber, "Modeling (seemingly) impossible models," *Comput. Graph.*, vol. 35, no. 3, pp. 632–638, 2011.
[18] Y. I. H. Parish and P. Müller, "Procedural modeling of cities," in *Proc. 28th Annu. Conf. Comput. Graph. Interactive Techn.*, 2001, pp. 301–308.
[19] P. Wonka, M. Wimmer, F. Sillion, and W. Ribarsky, "Instant architecture," *ACM Trans. Graph.*, vol. 22, no. 3, pp. 669–677, 2003.
[20] P. Müller, P. Wonka, S. Haegler, A. Ulmer, and L. Van Gool, "Procedural modeling of buildings," *ACM Trans. Graph.*, vol. 25, no. 3, pp. 614–623, 2006.
[21] M. Lipp, P. Wonka, and M. Wimmer, "Interactive visual editing of grammars for procedural architecture," *ACM Trans. Graph.*, vol. 27, no. 3, pp. 102:1–102:10, 2008.
[22] J. Danihelka, L. Kencl, and J. Zara, "Stateless generation of distributed virtual worlds," *Comput. Graph.*, vol. 44, pp. 33–44, 2014.
[23] M. Steinberger, M. Kenzel, B. Kainz, J. Mueller, P. Wonka, and D. Schmalstieg, "On-the-fly generation and rendering of infinite cities on the GPU," *Comput. Graph. Forum*, vol. 33, no. 2, pp. 105–114, 2014.
[24] G.-S. Wang. (2006). Hallucii [Online]. Available: http://www.imdb.com/title/tt1139791/
[25] Valve Corporation. (2007). Portal and Portal 2 [Online]. Available: http://www.valvesoftware.com/games/portal.html and http://www.valvesoftware.com/games/portal2.html
[26] Alexander Bruce. (2013). Antichamber [Online]. Available: http://www.antichamber-game.com/
[27] F. D. Davis, "Perceived usefulness, perceived ease of use, and user acceptance of information technology," *MIS Quart.*, vol. 13, no. 3, pp. 319–340, 1989.



Chi-Fu William Lai received the BEng degree in computer science (minor in mathematics) from the Hong Kong University of Science and Technology in 2008. He is currently working toward the PhD degree under the School of Computer Engineering, Nanyang Technological University, Singapore. He is a visiting scientist in the Singapore University of Technology and Design. His research interests include computer graphics, recreational modeling, optimization, and computer-aided design.

Sai-Kit Yeung received the BEng degree (First Class Honors) in computer engineering, the MPhil degree in bioengineering, and the PhD degree in electronic and computer engineering from the Hong Kong University of Science and Technology (HKUST) in 2003, 2005, and 2009, respectively. He is an assistant professor at the Singapore University of Technology and Design (SUTD), where he leads the Vision, Graphics and Computational Design (VGD) Group. He was also a visiting assistant professor at MIT and Stanford University. Before joining SUTD, he was a postdoctoral scholar in the Department of Mathematics at UCLA. He was also a visiting student at the Image Processing Research Group at UCLA in 2008 and at the Image Sciences Institute, University Medical Center Utrecht, the Netherlands, in 2007. His research expertise is in the areas of computer vision and computer graphics. He has served in the technical program committee of Pacific Graphics, Eurographics, SIGGRAPH Asia Poster and Technical Brief. His recent research focuses on scene acquisition, understanding and re-modeling. He is a member of the IEEE and the IEEE Computer Society.

Xiaoqi Yan received the BS degree in computer science and technology form Si Chuan University in 2009. She is currently working toward the PhD degree in the School of Computer Engineering at Nanyang Technological University in Singapore. Her research interests are in mathematical visualization, interactive interfaces, and computer graphics.



Chi-Wing Fu received the BSc and MPhil degrees in computer science and engineering from the Chinese University of Hong Kong, and the PhD degree in computer science from Indiana University Bloomington. His research interests include user interaction design in HCI and computer graphics. He has served in the technical program committee of SIGGRAPH Asia technical brief, SIGGRAPH Asia emerging technology, IEEE visualization, and ACM CHI Work-in-Progress, and is currently serving as associate editor of Computer Graphics Forum. His research interests include computer graphics, visualization, and human-computer interaction. He is a member of the IEEE.

Chi-Keung Tang received the MSc and PhD degrees in computer science from the University of Southern California, Los Angeles, in 1999 and 2000, respectively. He is currently an associate professor in the Chinese University of Hong Kong. Since 2000, he has been with the Department of Computer Science and Engineering at the Hong Kong University of Science and Technology, where he is currently a full professor. He was an adjunct researcher at the Visual Computing Group of Microsoft Research Asia. His research areas are computer vision, computer graphics, and human-computer interaction. He was an associate editor of *IEEE Transactions on Pattern Analysis and Machine Intelligence (TPAMI)*, and was on the editorial board of *International Journal of Computer Vision (IJCV)*. He served as an area chair for ACCV 2006, ICCV 2007, 2009, 2011, & 2015 and as a technical papers committee member for the inaugural SIGGRAPH Asia 2008, 2011, 2012, 2014 & 2015, and SIGGRAPH 2011 & 2012. He is a senior member of the IEEE and IEEE Computer Society.