Towards Real-time Procedural Scene Generation from a Truncated Icosidodecahedron

Francisco M. Urea[1]
and Alberto Sanchez[2]

[1] FranciscoMurea@gmail.com,
[2] Universidad Rey Juan Carlos
alberto.sanchez@urjc.es,

Abstract. The procedural method is cutting edge in gaming and virtual cities generation. This paper presents a novel technique for procedural real-time scene generation using a truncated icosidodecahedron as basic element and a custom physics system. This can generate a virtual museum in an interactive-way. For doing this, we have created a simple interface that enables creating a procedural scenario regarding the user-specified patterns, like number of pieces, connection mode, seed and constraints into the path. The scene is generated around three dimensional edges meaning a new point of view of hypermuseums. It has its own physics system to adapt the movement through the three dimensions of the scene. As a result, it allows the user to create a procedural scene in almost real-time where a user character can go over the 3D scene using a simple interactive interface.

Keywords: Procedural generation, 3D scene generation, real-time, hypermuseum, truncated icosidodecahedron

1 Introduction

Nowadays, the generation of virtual procedural scenes is growing importance in video games and virtual reality. The procedural generation is used in many areas, such as generating volumetric fire [FKM$^+$07], creating textures [RTB$^+$92] or piles of rock for building [PGGM09]. In fact, the building information modeling is a popular research area, e.g. with some works from Müller et al. [MWH$^+$06] and Patow [Pat12].

On the other hand, physical museums are trying to bring the art and history to the general public. The concept of hypermuseum [Sch04] was born to virtually recreate the interior of a museum. Thus, a hypermuseum is usually understood as an identical recreation of a physical museum. They are subject to the physics laws and have a typical path structure. This work presents a new point of view for hypermuseums unleashing the imagination. The idea is creating a hypermuseum regardless of the physical laws, following the Escher's picture "Relativity" or "Inception" movie. We use a truncated icosidodecahedron (TI) to create the virtual path rotating and connecting its different faces.

The procedural process generation is not used to be in real time because the idea is creating and saving something to be used later, but in this case, the hypermuseum is procedurally generated at the same time that it is gone over. Thus, this paper presents an almost real-time approach for procedural generation using a space voxelization.

The rest of the manuscript is organized as follows. In Section 2, several works related to our proposal are described. Section 3 presents the TI and explains the mathematics that have been used in the polyhedron and Sec. 4 shows the general idea of creating an hypermuseum with these pieces. Section 5 shows the process to generate a TI into the scene. Section 6 explains the path generation and the custom physics system. Section 7 explains the space division used for data storage and accelerating the generation process. Section 8 shows the evaluation of our proposal. Finally, Section 9 analyzes the conclusions and states the open issues of this work.

2 Related work

Some researches have previously created hypermuseums. For instance, [Sch98] generates an accurate representation of a museum using a WWW database. [WWWC04] generates virtual content in which visitors can interact on a display or via web. William Swartout et al. propose to create virtual guides using a natural language and human aspects [STA$^+$10].

Nevertheless, as far as we know, this proposal means the first time that a procedural process is being used to generate a hypermuseum. In any case, there are some works with similar procedural approaches. [STBB14] presents a complete survey with different procedural methods useful to generate features of virtual worlds, including buildings and entire cities that can be viewed as similar to hypermuseums. Different methods employ a combination of computational geometry algorithms similar to our proposal. Stefan Greuter et al. [GPSL03] describe a "pseudo infinite" city where several kinds of buildings are procedurally generated with different parameters. Julian Togelius et al. [TPY10] generate a map using a multi-objective evolutionary algorithm. [LSWW11] creates paths in a similar way to this work where a user can walk trough them. Finally, [CwCZ11] creates procedural elements using voxels as basic elements. Against these alternatives, this paper presents a novel technique towards real-time procedural scene generation.

3 Model description & breakdown

This section presents the figure used for generating the scene and explain the required mathematical basis to create it.

An icosidodecahedron is a Platonic polyhedron with twenty triangular and twelve pentagonal faces (see Fig. 1, first element). Every Platonic solid has a dual. The dual solid is the polyhedral element whose vertices are positioned in the center of each face of the first solid. The Platonic solid can be vertex-truncated by its dual when the ratios $d_{dodecahedron}/d_{icosahedron}$ (being d the radius between the faces and the center of polyhedron) get appropriate values. Their intersection gives Archimedean solids. In this case, an icosahedron and a dodecahedron are combined to obtain the icosidodecahedron.

Our figure called TI, presents a more complex geometry than an icosidodecahedron. We use the vertex truncation between an icosidodecahedron and a rhomb-triacontahedron. Figure 1 shows the obtained truncation. A complex polyhedra is used for connecting the faces by means of a connecting bridge allowing a user character to walk trough the faces. The face transitive duals of the series of the vertex-transitive polyhedra go from the rhomb-triacontahedron to the deltoid-hexecontahedron, passing through a continuous series of intermediate hexakis-icosahedra (also called disdyakis-triacontahedra). The following parameters are used for truncation:

$$d_{dodecahedron} = 1/\sqrt{1 + (1/\tau^2)} \tag{1}$$

$$d_{icosahedron} = \tau/\sqrt{3} \tag{2}$$

$$1/(1 + 1/\tau^2) \leq d_{rt} \leq 1 \tag{3}$$

where τ is the golden ratio and d_{rt} is the radius of a rhomb-triacontahedron.

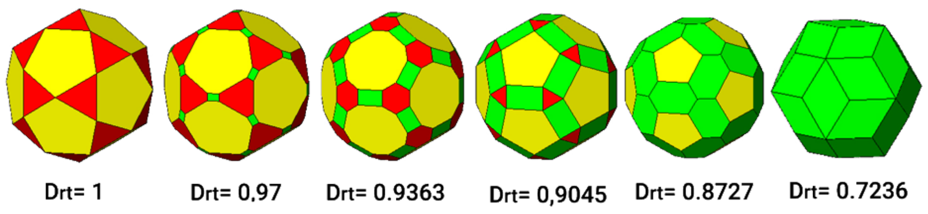


Fig. 1. Truncation process of an icosidodecahedron. A different polyhedron is obtained varying the ratio of the rhomb-triacontahedron.

Each triangular face of the TI is enumerated as follows. First, a face is randomly selected to be numbered with number one. For each face, three connected faces can be accessed numbering them in clockwise. Then, we get the second face and repeat this process. Finally, all local positions and rotations of the faces from the center of the icosidodecahedron are saved. For each face, there is another face with the same rotation and opposite position requiring only four data values in space position to position a face.

4 Truncated Icosidodecahedron Museum

The Truncated Icosidodecahedron Museum (TIM) consists on successively joining custom platforms using the geometry of the triangular faces of the TI. Only two custom platforms have been defined: i) looking outside of the TI and, ii) looking inside it. The platforms are built based on the icosidodecahedron's geometry, i.e. it generates a TI if twenty platforms were chosen. The user can select a series of parameters, such as the number of faces, the seed, connection mode between them and if the pattern can use less faces than the given. Introducing the same pattern and the same seed creates the same scene. A scene, where the user character has the capability for walking through it, is interactively generated.

Each platform of the TIM has three connection bridges as joints between neighbor faces of a TI. The faces neighbor to a face are called *border faces*. The connection between faces are used as paths. Some faces have not connected all their connecting bridges. These faces are called *limit faces* whereas all the used faces are called *useful faces*. We use the limit faces to generate new TIs from there to enlarge the TIM scenario. Figure 2 shows the differences between kinds of face.

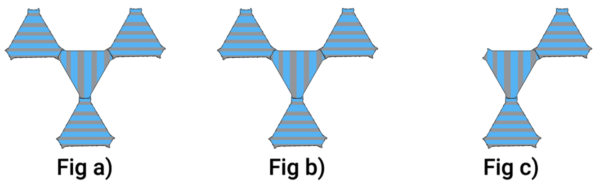


Fig. 2. The faces drawn in a) are useful faces. The faces with horizontal lines in b) are border faces of the face with vertical lines. The face with vertical lines in c) is a limit face.

5 Building a TI

This section explains the building of a TI and the problem found during its creation. Subsection 5.1 presents the method to get the useful faces. Subsection 5.2 explains the different kinds of face selection. Subsection 5.3 defines the generation of the TI into the virtual scene.

5.1 Getting of useful faces

Figure 3 shows the process to get the useful faces. The following buffers are required:

1. *All faces.* This list contains useful faces.
2. *Available faces.* This list contains the faces to get more faces, i.e. it stores the limit faces until the date. It requires a first face, named *inception face*, to generate the TI given in the constructor.
3. *Inaccessible faces.* This list contains the faces which we cannot use.

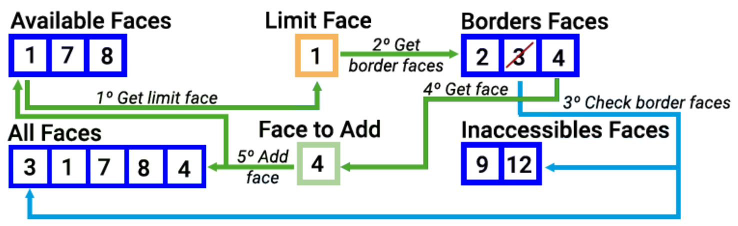


Fig. 3. Getting face algorithm loop.

Firstly, the inception face is added in “available faces” . Immediately a face is got from “available faces” as limit face. Then get the border faces to the limit face. If any of these border faces has already been used or it is contained on “inaccessible faces”, it is refused. Then, get one of the accessible border faces and add it to “all faces” and “available faces”. The process checks then the buffer of “available faces” removing all completed faces. This process is repeated until the size of “all faces” buffer is the same as the number of provided faces. If any face collides, the face is added into “inaccessible faces” buffer before the rebuilt process.

5.2 Connection mode

There are three different modes to connect a face with its border faces. All modes repeat the same process but the condition to select the next face to connect differs between them.

- **Random mode**. This mode takes a random face from the buffer of “available faces“. This face is used as limit face. Then get a random face from the border faces of this limit face. At the end, this face is checked and added to the buffers “all faces” and “available faces”.
- **Full mode**. This mode takes the first face of “available faces” as limit face. Then get the first face of its border faces. Once this face is checked, it is added to the buffers of “all faces” and “available faces”. After the clean process, if all border face of this limit face was added, this limit face was erased from “available faces” and take the first face of “available faces”as limit face. If the new limit face has not got border faces, the algorithm takes a random face of “ available faces” as limit face.
- **Linear mode**. At first, this mode randomly takes a face from “available faces”. Then get one border face. It is checked and added to “all faces” and “available faces”. Finally, instead of taking a face from “available faces”, take the last face added as limit face, and repeat the process. If this border face has not got border faces, the algorithm take one as limit face from “available faces”.

In every mode, the buffer of “available faces” is cleaned at the end of each iteration. This process consist on removing the completed faces.

5.3 Generation of a TI

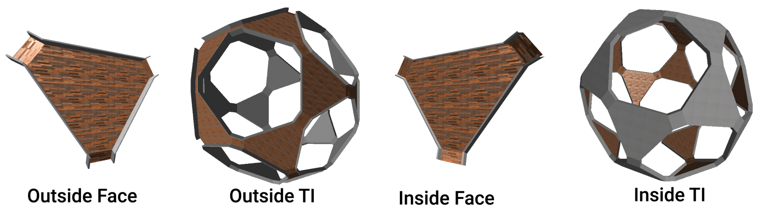


Fig. 4. Model of faces from TI and completed TIs

This section shows the generation of a TI in the virtual space. First of all, the algorithm requires the following parameters:

- **Inception face**. First platform to generate a TI. A platform is the representation of a face. In the first TI the algorithm gets a random number for selecting an inception face and zero to border face.
- **Blocked face**. Platform that cannot be used.
- **Buffer of "all faces"**.
- **Face mode**. Our proposal presents two different alternatives (see Figure 4). The first one (outside face) is used for walking outside the TI. In the other hand (inside face) is used for traveling inside the TI.

The process to generate a TI is represented in Algorithm 1. A platform is understood as the representation of a face.

Algorithm 1 Generation of TI faces in virtual space.

```
1: procedure PutFaces($pathList$)
2:     for all $faces$ do
3:         if $faceMode = 0$ then
4:             Create face with platform "FaceOutside"
5:         else
6:             Create face with platform "FaceInside"
7:         end if
8:         Add plattform to TI
9:         $ChangeLocalPosition(idFace, platform)$
10:        $ChangeLocalRotation(idFace, platform)$
11:    end for
12: end procedure
```

The algorithm instantiates the custom platform given the face mode and this process is repeated for each face from "all faces" buffer. Each platform requires the local position and rotation to place itself in the right position. These parameters are pre-calculated to make the process faster.

6 Path generation

To generate a path in the museum, firstly a TI is created regarding the user patterns as shown above. This is added into the “available TIs” buffer. Then, the algorithm follows the next steps: i) generating all possible TIs, ii) checking the new TI generated, iii) rebuilding the collided objects, and iv) repairing inaccessible paths. The algorithm stops when the “available TIs” buffer is empty. The sequence of creation a path from a TI is explained in next subsections.

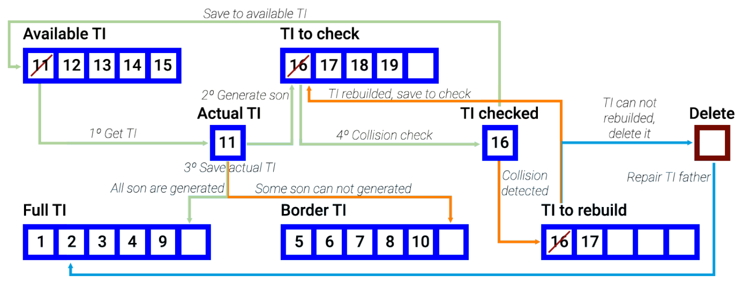


Fig. 5. Loop to generate a path by means of the union of TIs. Green line means the normal process. Orange line means the alternative case. Blue line is used when a TI is destroyed.

6.1 Generation of new TIs

The number of platforms varies depending on the user patterns. Some of these platforms have not connections with others. We use the limit platforms to generate new TIs and connect them. The parameters required to generate a new TI child are the number of faces of the new object, the connection mode and the condition if the path generated can be lower than the number of faces.

6.2 Union of icosidodecahedrons

First of all, the faces in available faces from TI are used as limit face to generate new unions. For each face from this buffer, its border faces are calculated. The border faces are needed for two reasons. First, the border face of the TI father is the dual face of the border face of the inception face from the new TI. It means that the border face of the new TI uses the connecting bridge to join both TIs. For this reason, this border face is removed from the border faces to avoid a collision. Second, the border faces are needed to know the where the connecting bridge is to position the new TI.

Next, the algorithm calculates the center position for the new object. The midpoint method is used for calculating the new center. As we explained in

Section 3, every face has an opposite symmetrical face. Thus, the algorithm uses the connecting bridge position between the two objects as the middle point. The algorithm applies the following equation to get the new center:

$$Center_{child} = Position_{bridge} * 2 - Center_{father} \tag{4}$$

The new generated object is added into the TI list to check the buffer. Then, if all unions of the TI have been generated, the object is saved into the “full TI” buffer. If some of the unions cannot generate a new object (Section 7 explains the reason), the object is added to the border list. This process is repeated for each TI into the “available TIs” buffer.

6.3 Collision Detection

We use a convex mesh around the platform (see Fig. 6) to avoid platform intersection. A ghost layer component is created for each platform. When a collision is detected, this component evaluates the latest created platform and selects which one has to be destroyed. At the end, if the platform is not destroyed, its ghost layer component is deleted. The algorithm has the following steps:

1. The collider detects a collision. If the collided platform has not a ghost layer component and it is not part of the TI father, it is marked to be destroyed.
2. If the collided platform has a ghost layer component, it is evaluated if its identifier is higher. If its identifier is lower, it is added to the collision list.
3. Before destroying the platform, the collision list is checked it. If there is any platform in the list, it is destroyed. If not, the platform remains without destroying.

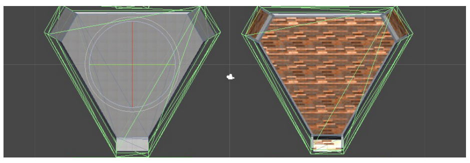


Fig. 6. Mesh collider

The collision detection process automatically starts when a platform is instantiated. If the platform has to be destroyed, a message is sent to the kernel process. This message contains the collided TI and the destroyed face. This face is saved into the “inaccessible faces” buffer and a new TI is rebuilt with this constraint.

6.4 Rebuilding TI

This process is run when a platform of a TI is destroyed. Then, the algorithm creates another TI avoiding the face corresponding to the platform taking into account the number of useful faces enabled by the user patterns. Nevertheless, if the inception face has collided, this TI is destroyed.

6.5 Repairing TI

The repair of a TI is required when some unions can not be generated, due to a TI destruction. This process consists on replacing the connecting bridge with a wall. When the platform is repaired, the collider is replaced with another one, which best matches into the mesh taking into account the connecting bridges destroyed.

6.6 Internal physics

A TI can be technically understood as a simplification of a sphere. This idea is used to generate an adaptable physics system. While the user walks trough the TI, the force is calculated using the Newton's universal gravitational constant (see Fig. 7). When the user walks outside the object, the force pulls the user to the center of the object, allowing to the user walks over the sphere surface. If the user walks inside, the force pushes the user from the center. The player system detects the mode and changes the direction of the force when it is necessary.

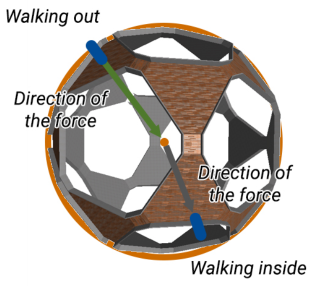


Fig. 7. Internal physics. The green arrow represents when the user walks outside the surface of the platform. The gray arrow represents when he travels inside the surface.

7 Space division

The representation space has been voxelized. Each voxel has the size of a TI storing the data that represents it. When the user character is moved, the cubes ot TIs, which have not to be represented, are saved with all its content to provide persistence. A map, where the key is the center of the voxel and the content is a list with all TI data, is used to accelerate the process.

In addition, the voxelization has other uses, like limiting the generation of TIs. The cube, where a TI is represented, is divided in three parts in all axis (see Fig. 8). Each one of the subcubes is a voxel to get faster memory accesses. When the user character walks and passes to other voxel, the system accesses memory only using the following nine cubes (instead of accessing the whole memory). This division also places the user in the center of the voxel seeing the existing objects in every direction.

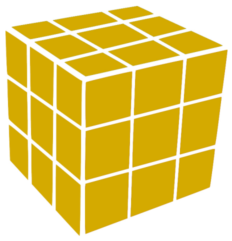


Fig. 8. Picture of division of representation space.

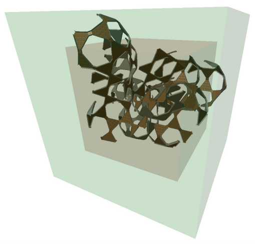


Fig. 9. Red cube is border margin to generate TI. Green cube is space to representation.

We create a margin border (see Fig. 9) to avoid collisions among the TIs generated in different time steps. The size of the margin border is half TI, making the generation of platforms outside the cube impossible.

8 Results

We test our proposal using a Intel i7-3630QM CPU laptop with a Nvida 660GTXM and 16Gb DDR3 RAM. The application has been developed by using the game engine Unity3D v4.6.2f1[3] to enable running it in any platform, like Windows, Mac, Linux, mobile or web. The user can walk over the scene specified regarding its own patterns by using the custom physics and sound steps, like if the user is inside of a museum. Figures 10 and 11 show examples of the scene created at the same time that is gone over. This work can be downloaded and tested in `https://dl.dropboxusercontent.com/u/697729/TIM/Index.html`.

To evaluate its performance, we have created a custom pattern and a specific tour to allow us to replicate the scene. We run the same testbed 7 times obtaining the Fig. 12.

Figure 12 is divided in six columns, which correspond to different steps of path generation. As we can see, the standard deviation is low, ensuring a similar behavior in different runnings. Regarding performance, the collision detection

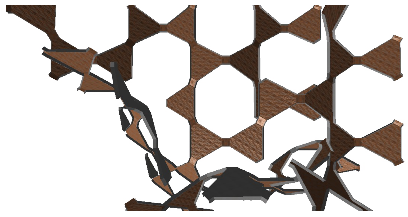


Fig. 10. Render in real time outside of scene.

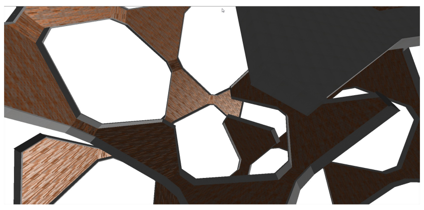
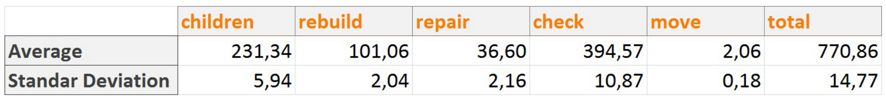


Fig. 11. Render inside the other scene.



	children	rebuild	repair	check	move	total
Average	231,34	101,06	36,60	394,57	2,06	770,86
Standar Deviation	5,94	2,04	2,16	10,87	0,18	14,77

Fig. 12. Average time in ms. and standard deviation for each step of the algorithm traveling around the scene with the following patterns: 1 face in random mode, 12 faces in full mode, 1 face in random mode.

phase requires most of the time. This is due to the Unity's collision system requires three internal loops to evaluate collision and destruction. Finally, we test the GPU vs CPU performance. We test different scenes, but the time using GPU is very similar to use CPU.

9 Conclusions and future work

This paper presents a novel technique to procedurally generate a hypermuseum from a TI. It could be adapted to generate any kind of scene. The user walks into an infinite procedural 3D scene created at the same time that it is gone over. This kind of generation presents a new point of view to understand hypermuseums. Additionally, we have created a panel to make user access easy to specific rooms. Thus, the user can select the room which would like to visit to teleport there.

As future work, we are planning to improve the collision detection system or create a custom one. Furthermore, we plan to adapt other kinds of figures to work, making more complex scenes and better adapted to the user needs, e.g to add the possibility of using for game scenarios. Finally, we plan to integrate procedural artworks.

Acknowledgments Thanks to Hector Zapata, CEO of Tetravol S.L., to propose this as work as a Master's Thesis and TI platforms.

References

[3D] Unity 3D. Unity 3d engine. `http://unity3d.com/`.

[CwCZ11] Juncheng Cui, Yang wai Chow, and Minjie Zhang. A voxel-based octree construction approach for procedural cave generation. *International Journal of Computer Science and Network Security*, pages 160–168, 2011.

[FKM$^+$07] Alfred R. Fuller, Hari Krishnan, Karim Mahrous, Bernd Hamann, and Kenneth I. Joy. Real-time procedural volumetric fire. In *Proceedings of the 2007 Symposium on Interactive 3D Graphics and Games*, I3D '07, pages 175–180, New York, NY, USA, 2007. ACM.

[GPSL03] Stefan Greuter, Jeremy Parker, Nigel Stewart, and Geoff Leach. Real-time procedural generation of 'pseudo infinite' cities. In *Proceedings of the 1st International Conference on Computer Graphics and Interactive Techniques in Australasia and South East Asia*, GRAPHITE '03, pages 87–ff, New York, NY, USA, 2003. ACM.

[LSWW11] M. Lipp, D. Scherzer, P. Wonka, and M. Wimmer. Interactive modeling of city layouts using layers of procedural content. *Computer Graphics Forum*, 30(2):345–354, 2011.

[MWH$^+$06] Pascal Müller, Peter Wonka, Simon Haegler, Andreas Ulmer, and Luc Van Gool. Procedural modeling of buildings. *ACM Trans. Graph.*, 25(3):614–623, July 2006.

[Pat12] G. Patow. User-friendly graph editing for procedural modeling of buildings. *Computer Graphics and Applications, IEEE*, 32(2):66–75, March 2012.

[PGGM09] A. Peytavie, E. Galin, J. Grosjean, and S. Merillou. Procedural generation of rock piles using aperiodic tiling. *Computer Graphics Forum*, 28(7):1801–1809, 2009.

[RTB$^+$92] John Rhoades, Greg Turk, Andrew Bell, Andrei State, Ulrich Neumann, and Amitabh Varshney. Real-time procedural textures. In *Proceedings of the 1992 Symposium on Interactive 3D Graphics*, I3D '92, pages 95–100, New York, NY, USA, 1992. ACM.

[Sch98] Werner Schweibenz. The" virtual museum": New perspectives for museums to present objects and information using the internet as a knowledge base and communication system. In *ISI*, pages 185–200, 1998.

[Sch04] Werner Schweibenz. Virtual museums. *The Development of Virtual Museums,ICOM News Magazine*, 3(3), 2004.

[STA$^+$10] William Swartout, David Traum, Ron Artstein, Dan Noren, Paul Debevec, Kerry Bronnenkant, Josh Williams, Anton Leuski, Shrikanth Narayanan, Diane Piepol, H. Chad Lane, Jacquelyn Morie, Priti Aggarwal, Matt Liewer, Jen-Yuan Chiang, Jillian Gerten, Selina Chu, and Kyle White. Ada and Grace: Toward Realistic and Engaging Virtual Museum Guides. In *Proceedings of the 10th International Conference on Intelligent Virtual Agents (IVA 2010)*, Philadelphia, PA, September 2010.

[STBB14] Ruben M. Smelik, Tim Tutenel, Rafael Bidarra, and Bedrich Benes. A survey on procedural modelling for virtual worlds. *Computer Graphics Forum*, 33(6):31–50, 2014.

[TPY10] Julian Togelius, Mike Preuss, and Georgios N. Yannakakis. Towards multiobjective procedural map generation. In *Proceedings of the 2010 Workshop on Procedural Content Generation in Games*, PCGames '10, pages 3:1–3:8, New York, NY, USA, 2010. ACM.

[WWWC04] Rafal Wojciechowski, Krzysztof Walczak, Martin White, and Wojciech Cellary. Building virtual and augmented reality museum exhibitions. In *Proceedings of the Ninth International Conference on 3D Web Technology*, Web3D '04, pages 135–144, New York, NY, USA, 2004. ACM.