CGI 2008 Conference Proceedings manuscript No.
(will be inserted by the editor)

Joost van Dongen

Interior Mapping

A new technique for rendering realistic buildings

Keywords interior · GPU shader · architecture

Abstract Interior Mapping is a new real-time shader technique that renders the interior of a building when looking at it from the outside, without the need to actually model or store this interior. With Interior Mapping, raycasting in the pixel shader is used to calculate the positions of floors and walls behind the windows. Buildings are modelled in the same way as without Interior Mapping and are rendered on the GPU. The number of rooms rendered does not influence the framerate or memory usage. The rooms are lit and textured and can have furniture and animated characters. The interiors require very little additional asset creation and no extra memory. Interior Mapping is especially useful for adding more depth and detail to buildings in games and other applications that are situated in large virtual cities.

1 Introduction

Many new games and virtual worlds feature large cities through which the player can move; take for example games like the *Grand Theft Auto*-series, *Crackdown* and *Superman Returns: The Video Game*, and the use of virtual 3D cities in online communities, simulations and virtual tourism. For a significant part, the graphical quality of these games is determined by how the buildings are rendered. These can have a certain amount of polygonal detail, depending on the desired size of the city and the dynamic loading system used, but the aforementioned applications all have in common that the cities are too large to model detailed buildings using geometry and still achieve interactive framerates while doing so. To deal with this, this paper introduces a new technique that significantly increases the graphical quality of buildings in real-time applications.

Utrecht University and Ronimo Games
E-mail: joost@ronimo-games.com

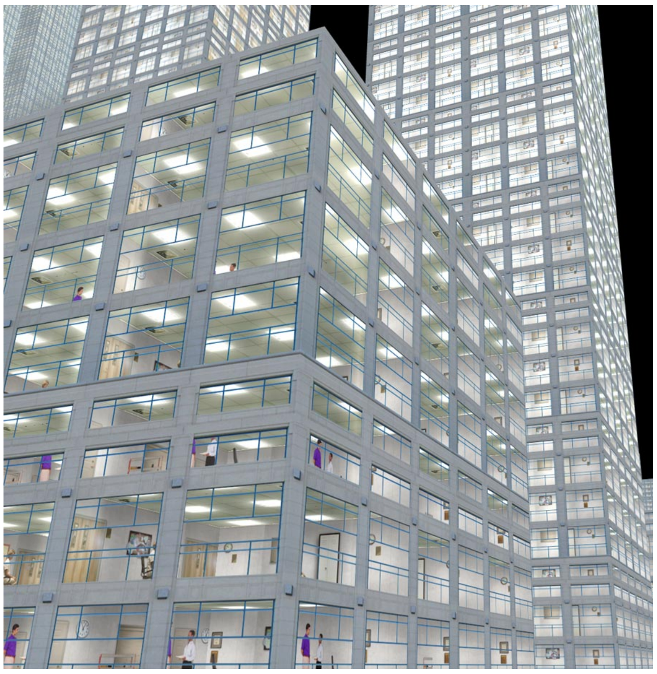


Fig. 1 Buildings with their interiors rendered with Interior Mapping. The geometry of the buildings consists of simple cubes only. Note how the interiors are textured and drawn perspectively correct.

Throughout the years, many techniques have been developed and used to visualise buildings as realistically and detailled as possible while still keeping framerates interactive. Geometry is often kept relatively simple, while details are added through the use of textures [4]. Originally, textures were only used to change the colours of surface elements. When adding details, this tends to look as if pictures of details have been glued on to a flat surface, instead of as if the details are really there. One technique that deals with the shading of details is bump mapping [1]. Unlike in the case of standard textures, with bump mapping, details are correctly lit with respect to the lighting in the scene. This can be extended through horizon mapping [12], to show the shadows of these details. In addition to bump mapping, normal mapping [7] is also often used, which stores the normals of each texture element, instead of their height. All the aforementioned techniques have in common that they do not correctly visualise the parallax effect when perspective changes. Displacement mapping [15] fixes this by rendering height-field surface details perspectively correct.

The texturing techniques mentioned above can be used to add details to buildings. There is also a number of techniques that have been developed specifically to quickly render more distant buildings and objects, as they are smaller in view and can thus be simplified. Displaying simpler geometry through level of detail [6] and lower resolution textures through mipmaps [18] are commonly used techniques for this. More rigorous is replacing distant objects by imposters [11] or turning complete groups of buildings into single blocks with height field shapes inside, which are rendered through the use of an extension to displacement mapping called block maps [5]. This way, hardly any polygons are needed, while relatively complex shapes can still be rendered with reduced detail.

None of the techniques above makes it possible to render the interiors of buildings. However, interiors are important for the graphical quality of a virtual city, as its buildings contain many windows through which interiors should be visible. But as interiors are often not required for gameplay, they are usually left out entirely, except in those instances where the interior can be entered by the player (see for example *Grand Theft Auto*). Adding the interiors of many buildings would require an enormous amount of polygons, which is not feasible when interactive framerates are required. Therefore, some other solution is required to render windows. Two simple techniques are used in most games and interactive applications today: either windows are fully reflective, not allowing a look inside, or some details very close to the window are drawn into the diffuse texture and the rest of the interior is left black.

One way to add perspectively correct interiors would be through the use of displacement mapping. If the displacement map is calculated separately from the exterior texture, then it could handle geometrically correct rooms. However, because displacement maps do not support texturing on surfaces that are perpendicular to the original polygonal surface, this would leave all walls with stretched coloured lines, except for the back wall, which could still use a real texture.

To fix this, textures can be added to all the walls of the interior through the use of indices from the displacement map into an extra texture, as is done with block maps [5]. Block maps cannot render furniture or characters inside the room, though, because they are limited to height field geometry. Also, if the viewer watches a room at the corner of a building, she would see a different room through the windows on the different walls. This would be awkward, as it should in fact be the same room, only seen through different windows. Finally, to vary the textures on the walls per room, knowledge of which wall is being rendered would be required and is not readily available with block maps.

Another technique that could render interiors, is generalised displacement mapping [17]. This technique could render interiors of any complexity, but has a number of drawbacks that make it ultimately unsuitable. The most important problem is that generalised displacement maps use too much memory. After heavy compression, a map with a resolution of 128x128 pixels still requires 4mb of memory, while a standard texture of the same dimension, as can be used with Interior Mapping, requires only 48kb of memory, or 8kb if DXT1 texture compression [2] is used. Also, with generalized displacement mapping it is not possible to separately animate a single character or object in the interior and walls cannot be varied independently of each other.

To deal with the problem of rendering the interiors of buildings, in this paper, Interior Mapping is introduced. This new technique solves the aforementioned issues with block maps and generalised displacement maps by directly rendering virtual walls through raycasting. The Interior Mapping algorithm knows which wall in which room it is rendering and can use this information to vary lighting and textures per room. Furniture and characters can be added through the use of furniture planes.

An added benefit of Interior Mapping is that it can easily be coupled with procedurally generated buildings, like in [13] and [19]. Most procedural city and building generation systems do not take interiors into account. Interior Mapping could easily be added to the buildings that these systems generate and works well even with buildings that have curved walls.

Although Interior Mapping is a technique that renders actual interiors, it does not need to model or store them in geometry. The walls of the rooms of the interior only exist as virtual geometry in the shader. For each pixel, the ray from the camera to the point on the building that is being rendered is intersected with the walls of the interior. Because the interior walls are regularly spaced, the ray can be collided in constant time, regardless of the number of rooms in a building or the number of buildings. Because it is shader-based, it is easy to turn Interior Mapping off for level of detail or lower quality rendering on older GPUs. Models do not need to be changed to make Interior Mapping possible, so the effect can quickly be added to cities that have already been created or are near completion.

Because the raycasting is done for each individual pixel that is rendered and takes the camera direction into account, the result is that perspectively correct rooms are rendered inside the building when looking at it from the outside. This can then be combined with a texture that stores where windows are, so that the interiors are only visible through the windows. By blending the interior's colour with a reflection map, both the interior and the reflections in the windows can be shown. See figure 1 for an example of what can be achieved using Interior Mapping. Note that the buildings here are modelled as single blocks.

Interior Mapping uses ray casting on the GPU, a research field that has quickly evolved since the introduction of programmable shaders. Here, a major trend is to turn the GPU into a full raytrace renderer [3], which means that the rasterizing functionality of the GPU is worked around to create a renderer that utilises the vector processing power of the GPU, but does not use the GPU's approach to rendering. Contrary to this approach however, Interior Mapping does not seek to *replace* rasterised GPU rendering, but actually to *enhance* it. Thus, it is more closely connected to techniques like displacement mapping and papers on topics like ambient occlusion [16]. Nonetheless, using the regularity of internal structures in the way that Interior Mapping does, does seem to be a novelty among existing GPU raycasting techniques.

Interior Mapping can be implemented in such a way, that the number of shader instructions is small enough to be able to calculate the effect within the 64 instructions allowed in pixel shader model 2.0 [10], and also add a diffuse and reflection map in the same pass. More effects can be added to the basic algorithm to create additional details.

The remainder of this paper is structured as follows: section 2 explains how Interior Mapping works. Section 3 analyses the performance of Interior Mapping and section 4 looks into a number of extensions to the basic technique.

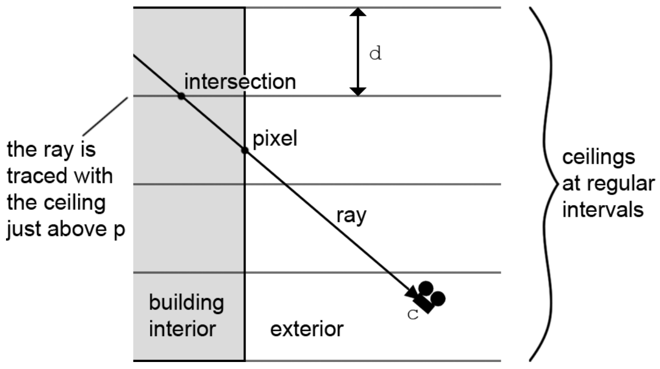
2 The algorithm

For ease of understanding, the simplest case is considered first: only ceilings and floors are being rendered. Walls, lighting and exterior textures will be added later.

2.1 Ceilings

For the Interior Mapping algorithm, the 3D space is considered to have ceilings at regular distances. Each ceiling is an infinite plane parallel to the XZ-plane. Intersecting a ray with a horizontal plane is a very simple thing to do, but the main point to Interior Mapping is to calculate efficiently which plane to use for a particular pixel.

When the pixel shader renders a pixel of a polygon to the screen, this pixel also has a position in the 3D world. This position is calculated in object space and the position of the camera is transformed into object space as well. When handling a ray from the camera to this pixel with a camera that is tilted upwards, the ceiling just above the pixel needs to be found. This can be done by taking the ceiling-function of the y-position of the pixel, divided by the distance d between ceilings. Afterwards, the height that has been found should be multiplied with d again to get the actual height of the ceiling. In Cg, this simply comes down to $ceil(y/d) \cdot d$. When the camera is tilted downwards, the ceiling (or actually the floor) that is one position lower must be used, so in that case the height is at $(ceil(y/d) - 1) \cdot d$.


d
intersection
pixel
the ray is traced with the ceiling just above p
ray
ceilings at regular intervals
building interior
exterior
c
Fig. 2 Visualisation in 2D of the variables in the formula above.

Now that the height of the ceiling is known, the ray from the camera to the pixel can be intersected with the ceiling to find the position where the ray hits the ceiling. This is visualised in figure 2.

The result is the position of the intersection, but what is needed is a colour for the pixel. This can easily be obtained by using the x- and z-coordinates of the intersection as the uv-coordinates for a texture read for the ceiling or the floor. To scale the texture to the desired size, the coordinates can be multiplied by some constant.

By performing all the calculations in object space, the walls can be rotated by rotating the space of the object itself. This way the walls in different buildings do not have to be parallel.

Note that an exactly horizontal ray will result in a division by zero. However, GPUs do not crash on this and the visual artefacts this results in are so rare in practice that this problem is negligible.

2.2 Walls

So far only horizontal planes have been considered. However, real interiors do not only have floors and ceilings, but also walls. These can be added by using exactly the same calculations as were used for the ceilings, but now with XY- and YZ-planes, instead of XZ-planes. The intersection of the ray is thus calculated with three different planes.

Of the resulting three intersections, the one closest to the camera will be used. To be able to use different textures for the ceilings, floors and walls, the texture corresponding to the closest intersecting plane is used.

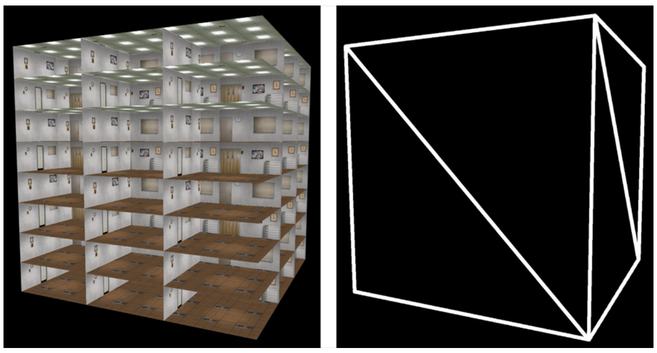


Fig. 3 The result of calculating ceilings, floors and walls with Interior Mapping. The geometry of this building consists of a single cube, as is shown in the wireframe image of the same building to the right.

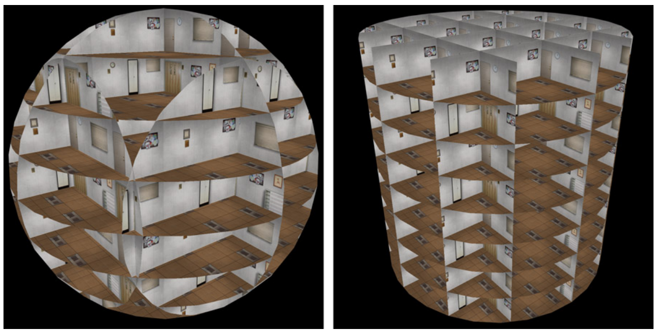


Fig. 4 Interior Mapping applied to a sphere and a cylinder.

This allows for different textures to be used for walls and ceilings, as can be seen in figure 3. The interior thus calculated works with curved geometry as well, as is exemplified in figure 4.

2.3 Combining with exterior textures

The basic Interior Mapping algorithm is now complete, but usually, it will be combined with a texture for the exterior of the building. This can be done by creating a diffuse texture that uses the alpha channel to store where windows are. If the alpha-value is 1, then the diffuse texture will be used; if it is 0, the colour calculated by the Interior Mapping algorithm will be used. This can be further refined by adding a reflection to the windows. The colour of the reflection will then be combined with the colour from the Interior Mapping, as can be seen in figure 5.

2.4 Furniture and animated characters

So far, the rooms in the interior have been completely empty. This works well when looking from a larger distance, but adding furniture and even animated characters to the interior would greatly improve their liveliness.

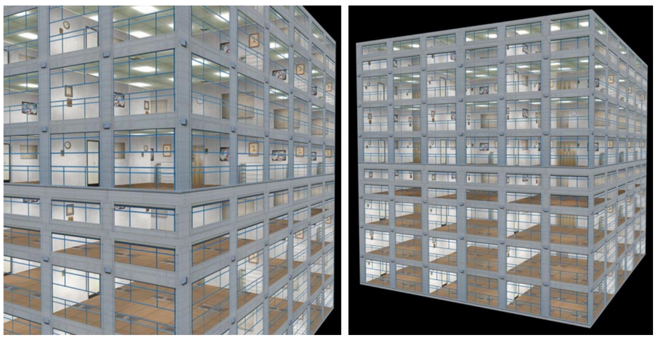


Fig. 5 Interior Mapping combined with an exterior texture and a reflection map. The reflection is made very subtle here to emphasise the effect of the Interior Mapping.

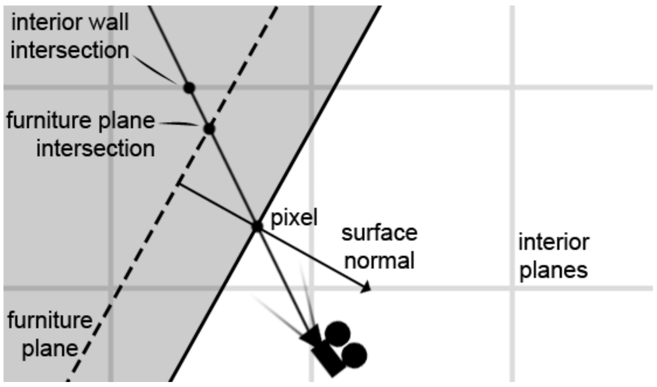


Fig. 6 The furniture plane is parallel to the actual surface of the object. The ray from the camera is intersected with both the furniture plane and the interior planes. In this example, the intersection of the ray with the furniture plane is closer than the intersection of the ray with the interior wall, so the furniture plane is visible.

Adding actual geometry to the interior is not possible without raycasting against large numbers of objects, so a different solution is needed here.

A technique that can add details inside the rooms is to add an extra plane that is parallel to the surface of the building, but displaced a fixed distance to the inside. Here, this plane is called the “furniture plane”. The furniture plane does not actually exist in the geometry and is defined in the pixel shader. It is intersected with the ray from the camera to the pixel and if the intersection is closer than any of the intersections of the interior walls, then the furniture is shown.

The furniture plane is intended to show things like furniture and characters, not to be a solid wall inside the building. Therefore, the alpha channel of the texture that is used for the furniture plane determines whether the colour of the furniture plane is discarded or not. This way, an object can be seen standing in the middle of a room, as if it were a billboard or sprite inside the building.

By using an animated texture, the objects on the furniture plane can actually move. Animated textures usually require a lot of storage, so only short animations

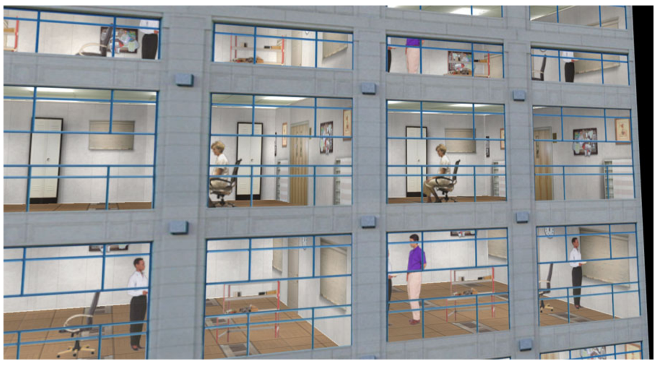


Fig. 7 An example of characters inside a room, made with a furniture plane.

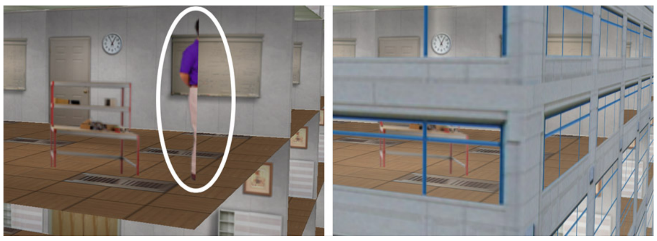


Fig. 8 In the left image, there is a seam at the corner of the building. This shows in that the centre character is cut-off. In the right image, this seam is hidden by a piece of exterior wall at the corner of the building, so that the seam can never be seen. Note that this seam appears here because the various sides of the building are oriented differently and thus have different furniture planes.

can be done this way. An example of this is a character reading a newspaper in a chair, turning over the pages once in a while. Through the use of render to texture [8], more complex animations could be shown on the furniture plane. This would require rendering separately an animated 3D character to a texture each frame and using this texture for the furniture plane.

Finally, note that unlike the interior planes, the furniture plane is not geometrically correct. If the building's surface is curved or has a corner, then the furniture plane will show distortions and even seams (see figure 8). The stronger the curvature and the further into the interior the furniture plane is, the stronger the distortion will be. For this reason, it is not advisable to add furniture planes to highly curved objects. Around the corners of buildings the solution is much simpler: avoid transparency in the exterior texture exactly at the corner. This also makes sense from an architectural point of view: most buildings do not have windows that span the building's corners anyway.

3 Experiments

To evaluate the effect of Interior Mapping for use in games and other real-time graphics applications, a test application has been written. This has been done in C++ using the OGRE 3D engine (`http://www.ogre3d.org`). All images in this paper were produced using this application. The reader can download the test application from the author's website (`http://interiormapping.oogst3d.net`).

A number of different versions of Interior Mapping have been implemented, so that they can be compared visually and regarding performance: with and without textures, with and without exteriors, with only ceilings and no walls, and with and without some of the extensions listed below. Through this test application, the different effects were analysed and their performance was measured, to see which implementation results in the best ratio between graphical quality and performance cost. The results are given below.

The other goal of this test application is to compare Interior Mapping to traditional polygonised buildings. Interior Mapping was compared to two different settings here: to the use of no interiors at all by making the windows only reflect, and to the use of polygons to create the interiors.

The test application measures performance by observing how many frames can be rendered in five seconds at a resolution of 1024 by 768 pixels. It runs automatically and writes the resulting frame-counts and the properties of the computer it ran on to a text-file. The test application was run on a computer with Windows XP, an AMD Athlon 2500+ processor, 1024mb RAM and an NVIDIA 6600GT GPU with 128mb video memory. A problem is that while running a test, Windows might decide to run some other process in the background. To keep this from cluttering the test results, the same test was run five times and any occurrences of oddly high numbers were not used in the results presented here. Furthermore, to verify that the test results are representative of other GPUs as well, the test was also run on thirteen other computers with various configurations. The results of these computers were similar to the main test computer, although faster GPUs of course achieved better framerates on all tests. The proportions between the framerates were roughly the same on all configurations, though. The complete test results of all fourteen computers can be found on the author's website.

3.1 Interior Mapping in comparison to polygonised interiors

It was expected that, if the scene is limited to a few buildings, Interior Mapping is slower than rendering interiors using actual geometry, because Interior Mapping uses a fairly complex pixel shader. When the number of buildings in the scene increases, then the performance of Interior Mapping should quickly overtake the polygonised interiors. In the test, an apartment building with 31 floors and 6 windows wide and long was used. The Interior Mapped version of this building is only 10 polygons, while the polygonised version takes 158 polygons.

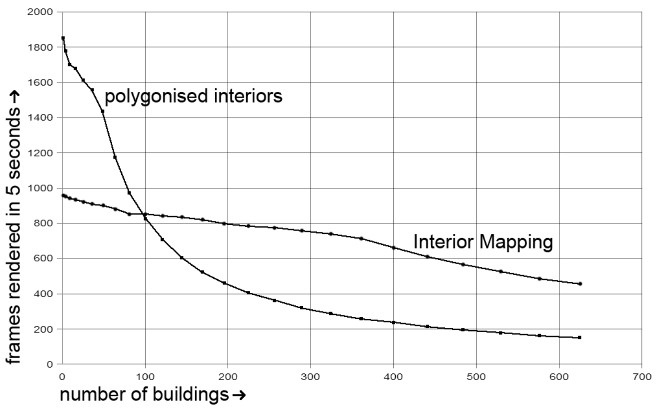


Fig. 9 Comparison between Interior Mapped buildings and polygonised buildings with interiors. The graph shows the number of frames rendered in 5 seconds for different numbers of buildings in the scene. More frames means a higher framerate and is thus better.

The Interior Mapped building can be rendered in a single draw call, while the polygonised version takes five draw calls. For the polygonised version, floors, ceilings and walls each use a different tiling texture and can therefore not be rendered in a single draw call. After that the reflection in the windows is first drawn and then the exterior walls. The version of Interior Mapping that was used here has four different textures for walls, ceilings and floors, while no lighting is calculated on the interiors.

For the Interior Mapped buildings, z-cull [9] is used. This is a technique where objects are first rendered only to the z-buffer and after that are rendered normally. Rendering to the z-buffer can be done very quickly, because it does not require the calculation of lighting, colour or complex effects like Interior Mapping. On the second pass, the z-buffer has already been filled with correct depths, so only the actually visible pixels are rendered and there is no overdraw at all. This greatly increases the performance of Interior Mapping, because overdraw means that Interior Mapping is performed several times on the same pixel on the screen. With z-cull, this never occurs. The actual performance increase of adding z-cull can vary widely from situation to situation, because it depends on the scene, the rendering order of the polygons and the camera angle. A test was performed with a single mesh that contained lots of buildings in a grid and a camera rotating around this scene. The results of this test show that in this specific case, the number of frames rendered in five seconds increases from 881 to 1265 when z-cull is used, an increase of 44%. This demonstrates that the performance of Interior Mapping can indeed greatly benefit from the use of z-cull. Because the framerate of the polygonised buildings is mainly dependent on the processing of the polygons, z-cull does not help there at all. In fact, all polygons need to be processed twice, so it even results in worse performance.

The graph in figure 9 shows how the number of frames rendered in five seconds changes when the number of buildings in the scene increases. As can be seen, the Interior Mapped buildings quickly overtake the polygonised ones. At 100 buildings, Interior Mapping already achieves a better framerate. This shows that the performance of the polygonised buildings is mainly dependent on the number of buildings, or in fact on the number of triangles, while the performance of Interior Mapping mainly depends on the number of Interior Mapped pixels that are actually drawn. The number of triangles is hardly relevant for Interior Mapping, because the polygon count is so low, that this is a negligible factor. At 500 buildings, Interior Mapping still only requires 5,000 polygons, whereas the polygonised buildings require 79,000 polygons. The reason why the Interior Mapped buildings do show a decrease in performance, is because the higher number of buildings still requires a higher number of draw calls. A draw call for a single object is an expensive operation, even if the object contains only 10 triangles and has no visible pixels.

3.2 Performance of the various versions of Interior Mapping

The previous paragraph showed that from a performance standpoint, Interior Mapping is better for large cities than creating interiors using polygons. However, there are a number of choices that have to be made when implementing Interior Mapping. Should it use only a single texture for the ceilings and leave the floors and walls blank, or should it use separate textures for the ceilings, floors and walls? And what is the effect of turning off Interior Mapping altogether?

Experiments show that using four separate textures for the floors, the ceilings and the two wall orientations results in the same performance as only texturing the ceilings and calculating lighting for the walls and floor. The performance of reading the colours of all walls and ceilings from a single texture was even worse. Apparently, the extra instructions required to calculate which of the texture coordinates to use to read from the single texture are less efficient than just reading from four different textures. This is in fact an advantage, as using four different textures allows artists to create much more graphically interesting rooms. The conclusion is that using four different textures is not only the best choice when graphical quality is required, but also when performance is required.

As expected, Interior Mapping is much more expensive than buildings that only have reflecting windows. A material that has a diffuse texture and reflections in the windows, which is the basic material for traditional buildings, renders 5,100 frames in 5 seconds. The same material with Interior Mapping renders only 999 frames. However, Interior Mapping adds a lot of detail to the buildings and is still easily fast enough to be applied in real-time applications.

Another important conclusion is that the resolution of the textures for the interiors does not influence performance much. When using four textures with a resolution of 256 by 256 pixels, 999 frames were rendered in five seconds. When switching to textures of 64 by 64 pixels, 1032 frames were rendered in the same time and at 16 by 16 pixels, 1053 frames were rendered. Decreasing the texture resolution this way therefore only resulted in a 5% performance increase, while the detail inside the rooms significantly decreased.

Another test analysed the performance with different numbers of rooms in a single building. The same test was run several times, while the number of rooms in the building gradually increased from 1000 to 4,000,000. As expected, this does not influence the performance at all. The framerate did not decrease and only showed a negligible random variation between tests.

4 Extensions

Although Interior Mapping adds a lot of detail to a building, the interiors are very repetitive. In this section three extensions are discussed that add more variety to the different rooms.

4.1 Varying lighting per room

In a real city at night, some rooms will have the lights turned on, while other rooms are in the dark. This effect can be added to the basic Interior Mapping algorithm through the use of a 3D noise function. The building has a variable c between 0 and 1 that stores the probability that the lights in a room are on. For each room, a 3D noise-function is used to retrieve a random variable r between 0 and 1. The room is only lit if $r < c$. This way a room is either lit or unlit. If c is slowly increased, then gradually more and more rooms in the city will become lit, as can be seen in figure 10.

4.2 Varying textures per room

In the simplest approach, the texture for the interior walls will contain an image of the wall of a single room. This image is used for all the rooms, so they all look exactly the same. However, through the use of a texture atlas [14], it is possible to have different textures for each room. A texture atlas is a single texture that contains several different textures, in this case several variations to a room texture. For each room, one texture is randomly chosen from the texture atlas. An example of using a texture atlas can is shown in figure 11.

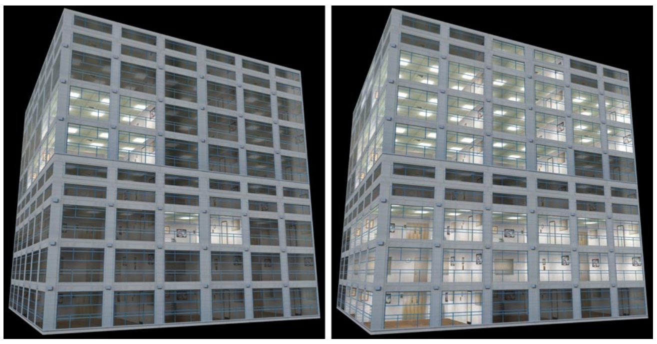


Fig. 10 The variable c determines the probability that the lights in a room are on. By increasing this variable, the interior of the building goes from completely dark to completely light.

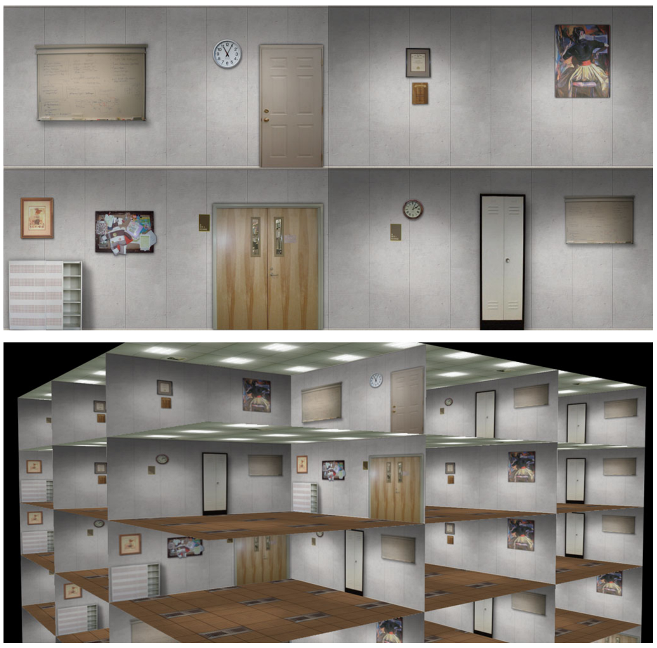


Fig. 11 A texture atlas containing four different room textures. Below it the result of randomly choosing one texture for each wall of each room.

4.3 Varying room sizes

So far, all rooms have been rendered with the exact same size. Although buildings with such a floorplan do exist, many buildings feature more variation in the sizes of their rooms. Unfortunately, if walls can have any position, it is unclear how to determine the interior wall closest to a pixel in constant time without a large space overhead. However, we can still achieve constant time if we limit how far the walls can be displaced from their position in the grid. In figure 12 an example is shown where walls can only be displaced from their original position by a maximum distance. Because walls can never be displaced further than the size of a room, only two walls need to be checked to find the wall closest to a

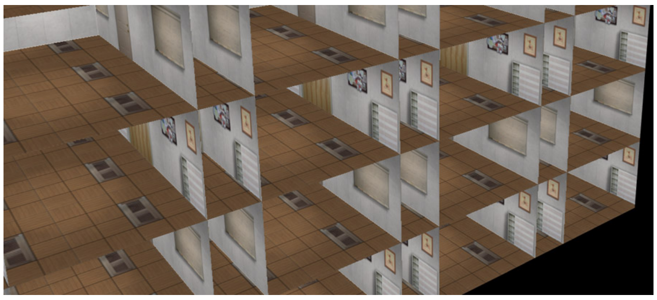


Fig. 12 Walls with varying distances between each other are possible with Interior Mapping as well.

certain pixel. This way, performance remains high, while more variation can be added to the rooms.

5 Conclusion

Interior Mapping has great potential for usage in the coming generation of computer games and other applications situated in virtual cities. Buildings can gain a lot of depth by adding interiors to them. Interior Mapping requires little extra work from the artists and only the extra storage space needed for the textures. It is efficient enough to be applied to games and virtual worlds for the Xbox 360, Playstation 3 and current and coming generations of PC games. Interiors made with Interior Mapping have complexity linear in the number of pixels on the screen, but constant in the number of buildings, windows, ceilings, floors and walls. As distant buildings occupy less pixels on the screen, this generates an automatic form of level of detail that greatly reduces the cost of rendering a building in the distance, whereas polygonised interiors would require the creation of more low-polygon versions of the same building to create level of detail. Interior Mapping is a great addition to any computer game that features large numbers of buildings.

Acknowledgements The author would like to thank the following people for valuable input and/or running the test application on their computers: prof. Dr. Mark H. Overmars, Thomas C. van Dijk, Ralph Rademakers, Erik van Dongen and the OGRE 3D community.

References

1. Blinn J. F.: Simulation Of Wrinkled Surfaces. ACM SIGGRAPH Computer Graphics **volume 12, issue 3**, 286-292 (1978)
2. Brown, P.: S3 Texture Compression. NVIDIA Corporation, November (2001)
3. Carr, N. A., Hall, J. D., Hart, J. C.: The Ray Engine. ACM SIGGRAPH/Eurographics Conference on Graphics Hardware, Saarbrucken, Germany, 37-46 (2002)
4. Catmull, E. E.: Computer Display Of Curved Surfaces. In: IEEE Conference on Computer Graphics, Pattern Recognition, and Data Structure, Los Angeles, May, 11-17 (1975)
5. Cignoni, P., Di Benedetto, M., Ganovelli, F., Gobbetti, E., Marton, F., Scopigno, R.: Raycasted Blockmaps For Large Urban Model Visualisation. Computer Graphics Forum **Volume 26, number 3**, 405-413 (2007)
6. Clark, J.: Hierarchical Geometric Models For Visible Surface Algorithms. Communications of the ACM **19(10)**, 547-554 (1976)
7. Fournier, A.: Normal Distribution Functions And Multiple Surfaces. Graphics Interface '92 Workshop on Local Illumination, May, 45-52 (1992)
8. Green, S.: The OpenGL Framebuffer Object Extension. Game Developers Conference (2005)
9. Kilgariff, E., Fernando, R.: The GeForce 6 Series GPU Architecture. GPU Gems 2, 471-491 (2005)
10. Kirk, D., et al.: Cg User's Manual. Nvidia (2004)
11. Maciel, P.W.C., Shirley, P.: Visual Navigation Of Large Environments Using Textured Clusters. SI3D, 95-102, 211 (1995)
12. Max, N. L.: Shadows For Bump-Mapped Surfaces. Proceedings of Computer Graphics Tokyo '86 on Advanced Computer Graphics, Tokyo, Japan, 145-156 (1986)
13. Muller, P., Wonka, P., Haegler, S., Ulmer, A., van Gool, L.: Procedural Modeling of Buildings. ACM Transactions on Graphics **Volume 25, Issue 3**, 614-623 (2006)
14. NVIDIA: Improve Batching Using Texture Atlases. SDK Whitepaper, NVIDIA, july (2004)
15. Oliveira, M. M., Bishop, G., McAllister, D.: Relief texture mapping. Conference On Computer Graphics And Interactive Techniques, 359-368 (2000)
16. Pharr, M., Green, S.: Ambient Occlusion. In book: GPU Gems, Addison-Wesley Professional (2004)
17. Wang, X., Tong, X., Lin, S., Hu, S. Guo, B., Shum, H. Y.: Generalized Displacement Maps. Eurographics Symposium On Rendering, 227-233 (2004)
18. Williams, L.: Pyramidal Parametrics. In Peter Tanner (ed.), Computer Graphics (SIGGRAPH 83 Conference Proceedings) **Volume 17**, 1-11 (1983)
19. Wonka, P., Wimmer, M. Sillion, F. Ribarsky, W.: Instant Architecture. ACM Transactions On Graphics **Volume 22, Issue 3**, 669-677 (2003)