Project-Report: Improved rendering performance using texture atlases

Wolfgang Knecht[*] Oliver Mattausch[†]
Vienna University of Technology, Institute of Computer Graphics and Algorithms

Abstract

Rendering huge scenes like cities means high state change costs for the graphics API because there are very much objects with different textures and very much Vertex Buffer Objects (VBO) are needed to render the scene. The goal of this project was to merge together some geometry to reduce the number of state changes. This implies that also the different textures have to be merged. Texture atlases are perfect for merging textures. Small textures get put together to one (or more if necessary) larger textures.

Because we already do some 3D file format conversion in this project we do some additional improvements on the raw .obj files. The raw data are not indexed so there are many vertices which exist several times. We remove such vertices and use indexed vertices instead. The raw data contain hard edges only. There are no averaged normals for curved surfaces. So we average normals of vertices of the same position when the angle between the normalvectors is below a threshold to get smoothed surfaces.

1 Introduction

This project is an addition to the 'Friendly Occlusion Culling' program from Oliver Mattausch which demonstrates the CHC++ algorithm [Mattausch et al. 2008]. The demo uses it's own 3D file format, the .dem format. There was an existing conversion tool to build such .dem files out of multiple .obj and .mtl files.

This converter did no optimization on the raw vertex and material data so the goal of the project was to add some optimizations to the converter. On the one hand we improved the optical result by averaging similar normalvectors to get smooth curved surfaces and on the other hand we tried to speed up the rendering process by creating indexed vertices and by merging groups of small objects together to one larger object which reduces state changes of the graphics API. Because every small object can have an individual texture the textures of the grouped objects had to be merged as well.

So the main challange of this project was to create correct texture atlases out of these individual textures. This includes remapping the texture coordinates and doing the repeat texture adressing mode by hand.

2 Implementation

We tried to do the new converter in a way that the .dem files which have been generated from the old converter can be read from the demo engine. So we write to the first some bytes of the new files the string 'format02' to identify if the file should be interpreted as the old or the new format.

[*]e-mail: wolfgang.knecht@aon.at
[†]e-mail:matt@cg.tuwien.ac.at

2.1 Indexed vertices

Creating indexed vertices out of the raw data from the .obj files which are not indexed is no big deal. We read the vertex positions, their texture coordinates and normalvectors after each other and compare them with all previously read combinations. If the same vertex is found we refere to this vertex in the index list. Otherwise a new vertex gets created.

2.2 Averaging normalvectors

After creating the indexed vertices we search for vertices which have nearly the same normalvectors. Therefor we investigate if the angle between normalvectors of vertices which are at the same position is smaller than a given threshhold (30 degrees for example). So we get groups of vertices with similar normalvectors. For each group we calculate the average of the normalvectors. The result are smoothed surfaces. In figure 1 the difference between smoothed and hard edges can be seen.

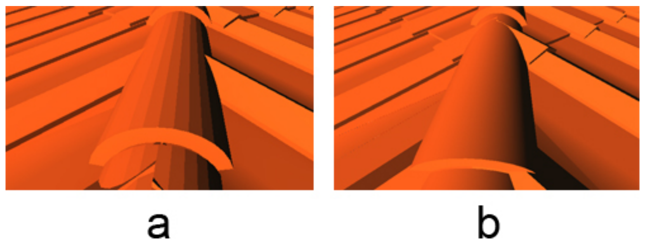


Figure 1: Hard (a) and smoothed edges (b)

2.3 Merging geometry

Because the scenes exist of very much small objects many VBOs are needed to render the scene. As mentioned above the VBO count has to be reduced to speed up the rendering process. But it is not possible to put the whole static scene in one VBO because then the culling algorithm CHC++ would have no effect at all. The aim is to reduce the object count but still have enough objects to cull some of them. For that we merge the whole scene together and use a bounding volume hierachie (BVH) [MacDonald and Booth 1990] which splits the scene using the surface area heuristic (SAH). The bounding volumes get split recursively until the vertex count is less then a given threshold in each leaf of the hierachy.

2.4 Creating texture atlases

A texture atlas is a collection of multiple textures in one texture. Figure 2 shows an example of a texture atlas. After merging the geometry there are many objects with different textures put together to one single object. So also the textures have to be merged. Texture atlases can be used for exactly this purpose.

We could now create one texture atlas for each leaf in the BVH. But that has the drawback that we have a lot of redundant textures because a texture could be used in one leaf and in an other leaf as well. Another problem is that our texture atlases do not have an infinite size. So it could be that all the textures of one leaf do not fit in one texture atlas. In this case we just split the leaf with the SAH again and again until all textures of the leaf fit into one texture atlas.

To reduce the redundancy of textures mentioned above we try to reuse texture atlases as much as possible. Therefore we look for the atlas which fits best to the one we need. If there are some textures missing in the best fitting atlas we try to add this textures to the atlas. If this fails because there are not enough free regions for the textures in the atlas we look for the next best fitting atlas an go on the same way.

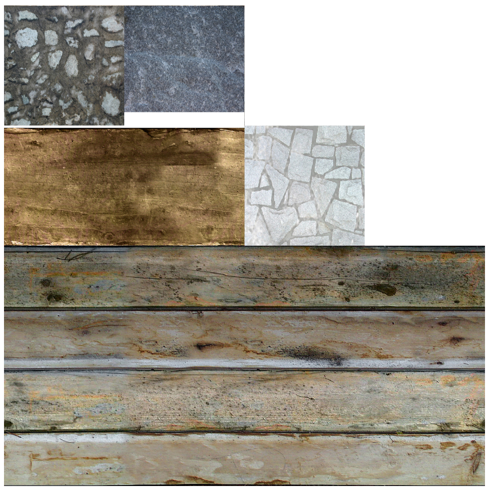


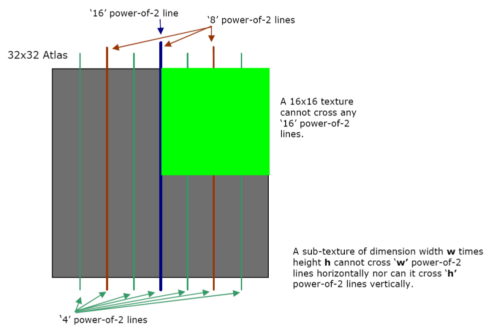
Figure 2: Texture atlas

Our strategy to arrange the textures in the atlas works as follows:

- sort the textures by there size and start with the largest textures
- divide the rest of the texture atlas into rectangular regions and remember them
- do the same with the next texture, try to put the texture in the smallest free regions at first

2.4.1 Correct mip mapping

When creating a texture atlas the most important thing is to take care about mip mapping. At first all the textures must have power of two dimensions to make correct mip maps possible. The next thing is that textues must not cross lines which are a multiple of their dimensions because otherwise the texles result in a mix of different textures even in height mip map levels. Figure 3 shows some Power-of-2 lines which must not be crossed by textures with the corresponding dimensions [NVIDIA-Whitepaper 2004].


'16' power-of-2 line
'8' power-of-2 lines
32x32 Atlas
A 16x16 texture cannot cross any '16' power-of-2 lines.
A sub-texture of dimension width **w** times height **h** cannot cross '**w**' power-of-2 lines horizontally nor can it cross '**h**' power-of-2 lines vertically.
'4' power-of-2 lines
Figure 3: 16, 8, and 4 Power-of-2 Lines for a 32x32 Atlas

Another thing we have to take care about is texturefiltering. We do not want to use nearest neighbor filtering, instead we want at least linear interpolation. That means lookups at the borders of textures will smear into neighboring textures which creates ugly seams. As a solution we create a border around our textures to avoid this artefacts.

The border has to be $2^{mipmaplevel-where-the-texture-has-no-seams}$ pixels width and the colors are taken from the opposite row of the textures border. That means a lot of additonal texturememory is needed but the result looks much better than without these borders.

2.4.2 Displace UV texture coordinates

Of course the texture coordinates have to be displaced to do correct lookups in a texture atlas. That's very easy because we just have to consider that our texture was from the coordinates 0.0 to 1.0 before and in the atlas now it's from an other start coordiantes to an other end coordinates. We can calculate an offset and a factor out of this input and displace all texture coordinates with them.

2.4.3 Texture adressing

We want to allow the texture adressing mode REPEAT in the demonstration program. But that doesn't work with a texture atlas. Instead of repeating the texture texels from other textures will be returned by the lookups. To avoid that we do the repeating by hand in a shader.

3 How to use the converter

In the 'FriendlyCulling' - solution there is the project 'Converter'. In this project the input and output files have to be specified as arguments and in the 'main' method of 'main.cpp' the number of obj files has to be set ('numFiles').

In the 'ObjConverter2.h' are the new variables for different conversion settings:

format02 Set this define to 1 if you want to convert to the new optimized version. Uncomment if you want to convert to the old version without texture atlases.

maxVertices While creating the geometrygroups a leaf in the BVH does not get split anymore when the vertex count in this leaf is less than maxVertices.

maxAtlasDim Specifies the dimensons of the atlases.

textureBorder Describes the border width of each texture in the atlases ($2^{mipmaplevel-where-the-texture-has-no-seams}$)

maxSmoothAngle Edges get smoothed when the angles between the normalvectors are below this threshold.

4 Results

Now we want to compare the framerates from the scene without optimization to the framerate of the scene with the optimization described above. We did a walkthrough through a vienna city model on a NVIDIA GeForce GTX 285 with several different optimization settings. The settings differ in the size of the texture atlases and the count of vertices in a merged geometry group.

Figure 4 shows the result of this performance analysis using a 1024x768 viewport. The first value in the legend of the diagram describes the texture atlas dimensions and the second one stands for the vertex count. The red graph is the unoptimized scene with no texture atlas at all. As we can see most of the time the optimized variants perform better. In our case an atlas dimenson of 4096x4096 and a vertex count of 5000 vertices per geometry group was a good choice. But it depends on the scene and the best setting has to be found empirically.

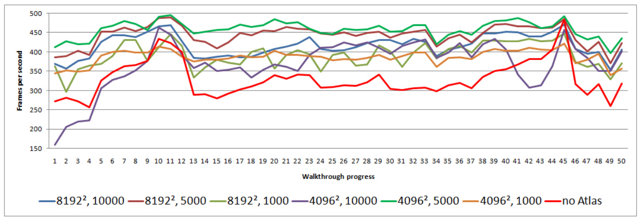


Figure 4: Performance analysis with a 1024x768 viewport

Figure 5 shows the same analysis with a 640x480 viewport. The smaller viewport improves the framerates in all variants of course. But the average difference of the best atlas variant and the variant without an atlas is about 0.8 milliseconds per frame in both cases. It seems the improvement doesn't scale with the viewport size. That makes sence because the optimization does not affect the fragment part a lot.

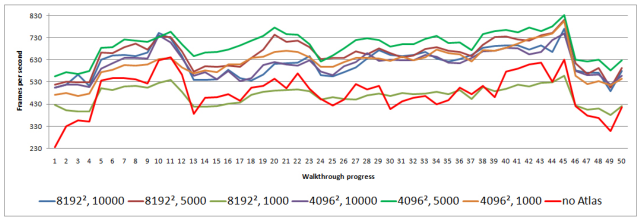


Figure 5: Performance analysis with a 640x480 viewport

The walkthrough for the performance analysis above was through the streets of the vienna model where only small parts of the whole city were visible. So we placed the camera above the roofs of the city to get an overview where we could see a lot of the city. In Figure 6 is the result of this test. Also in this case the optimized variants perform better than the variant without an atlas.

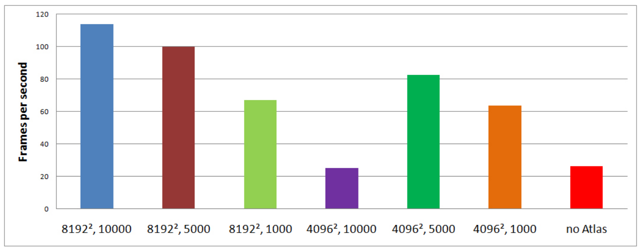


Figure 6: Performance analysis - large area overview

References

MacDonald, D. J., and Booth, K. S. 1990. Heuristics for ray tracing using space subdivision. *Vis. Comput. 6*, 3, 153–166.

Mattausch, O., Bittner, J., and Wimmer, M., 2008. Chc++: Coherent hierarchical culling revisited, Apr.

NVIDIA-Whitepaper, 2004. Improve batching using texture atlases, July.