Real-Time Rendering VU, 186.140
Demo Documentation

Stefan Doppelbauer

0656930, 033 532

stefan.doppelbauer@gmx.at

Martin Cerman

0625040, 066 932

mcerman@prip.tuwien.ac.at

January 18, 2013

Abstract

This is the documentation for the second assignment of the lecture with exercise named Real-Time Rendering VU. It contains a short description of the structure of our demo, a list of implemented effects, as well as a couple of problems we ran into.

1 “City Walk”

1.1 Structure

The structure of our demo code can be seen in the Figure below:

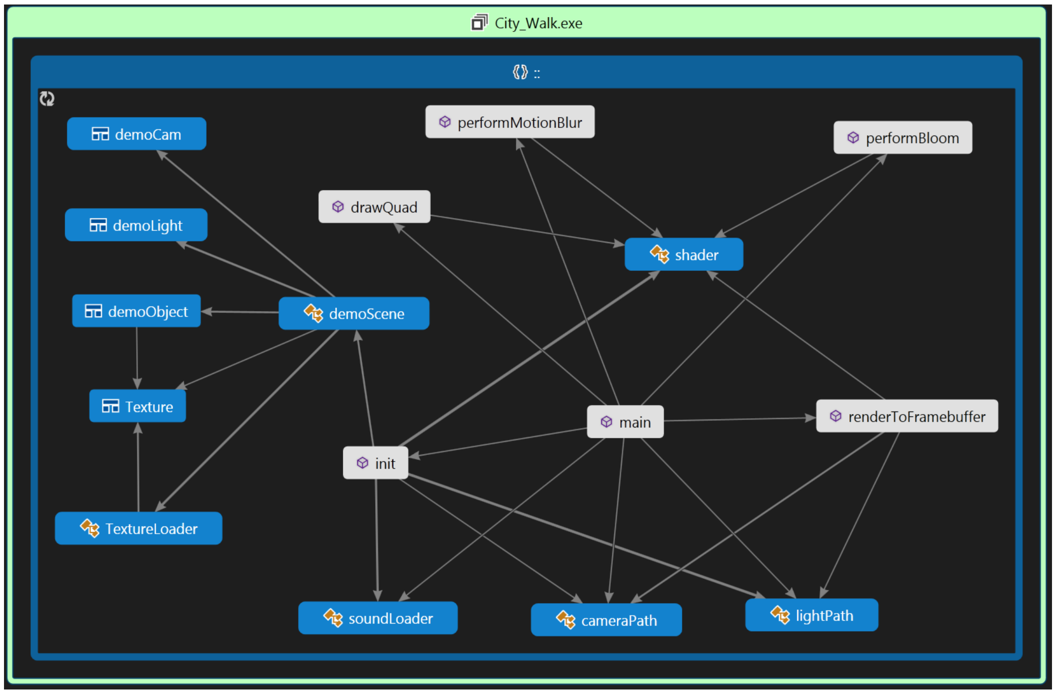

Figure 1: Structure of our demo code.

As can be seen, we use six classes, each of them abstracting a certain logical object. The class *demoScene* contains data about our demo scene and a method to load it from a COLLADA file. More in depth, it stores the necessary VAOs of our objects in the scene and data concerning the light and camera. Further it creates a *TextureLoader* object, which is responsible for loading textures and binding it to our objects. The class *soundLoader*, as the name also hints, is responsible for loading and managing the sounds, in particular the background music. The classes *cameraPath* and *lightPath* control the movement of the camera and light, so that our effects are better visible. These paths are modeled with Catmull-Rom splines, which are included in the GLM library. The sixth class is named *shader* and it is responsible for our vertex and fragment shaders.

1.2 Effect List

Below is the list of effects, that have been implemented in the current release version:

- Bloom [1] - Martin Cerman
- Motion Blur [2] - Martin Cerman
- Parallax Mapping with Offset Limiting [3] - Martin Cerman
- Scene Transition Effect (own idea) - Martin Cerman

1.2.1 Bloom

The bloom effect has been implemented the way as it was presented in the lectures and in the GPU Gems 1 [1] book, adding a small variation to the thresholding part. This effect is implemented in three passes, in which each pass draws its result into a separate texture inside a FBO. The first pass converts the existing color texture to a grayscale map and then thresholds it. All brightness values above a certain threshold are kept, the rest is set to zero. Using the grayscale conversion procedure before thresholding, we decouple the color from the brightness, resulting in a subjectively better effect. In the second and third pass, the thresholded texture is blurred horizontally and vertically using a 7x7 Gaussian kernel. We decided to implement this blurring in two passes, because it reduces the amount of texture lookups and thus increases the performance. We also decided not to use smaller images generated by mipmaps, so that the fine details given by the parallax mapping effect are not destroyed. The bloom effect is shown in Figure 2 and 3.

1.2.2 Motion Blur

The motion blur effect has been implemented as it is shown in [2]. This effect is drawn directly into a seperate texture in one pass. Herefore we supply the fragment shader with the original color and depth texture and the modelviewprojection matrix of the previous and current frame. Using these MVP matrices, it is possible to compute a velocity map, which defines for each fragment how far it moved within one frame. The next step is to accumulate a certain number of color samples, given by texture lookups along this velocity vector. The motion blur effect can be seen in Figures 4 and 5.

1.2.3 Parallax Mapping with Offset Limiting

The implementation of this effect requires not only direct processing in the shaders, but some preprocessing as well. The problem here is, that we will be working with textures mapped onto uneven surfaces, and herefore we need to be working in the tangent space of the surface. Therefore in the preprocessing step, we compute the tangents for each vertex. This is done manually in *demoScene.cpp*, while loading the object from the COLLADA file. First we extract the vertices, normal and texture coordinates of each face to compute the tangents. Next we find indices of identical vertices and use them to average tangents, that are within a certain angle. This way we dont get too hard edges at surfaces, that should normally be smooth. These tangents are then passed to the vertex shader, where we transform the light- and view vector into our tangent space (orthonormal basis given by the normal, tangent and bitangent). In the fragment shader we extract the height for each fragment from the height texture and compute the texture offset. We found it a bit hard to find the matching scaling and bias in the height computation formula, but in the end it worked out just fine. The results of the offset limited parallax mapping effect can be seen in Figures 6 and 7.

1.2.4 Scene Transition Effect

This effect was inspired by several modern day movies and games. The implementation is very simple and consists of a sequence of texture offsets in the horizontal direction. Here the red channel stays always the same, the green channel is offset by n pixels and the blue channel is offset by $2n$ pixels. All in all its a simple effect, but looks subjectively good, when switching between scenes.

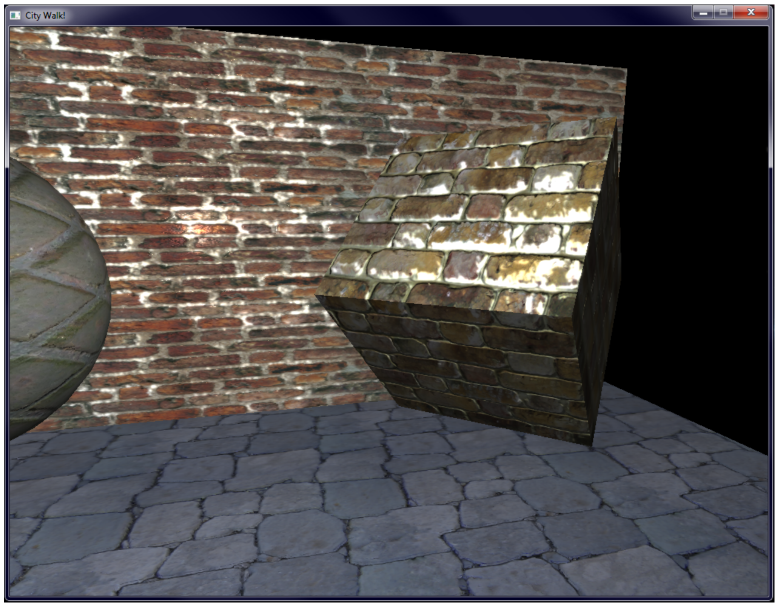

Figure 2: Bloom effect.

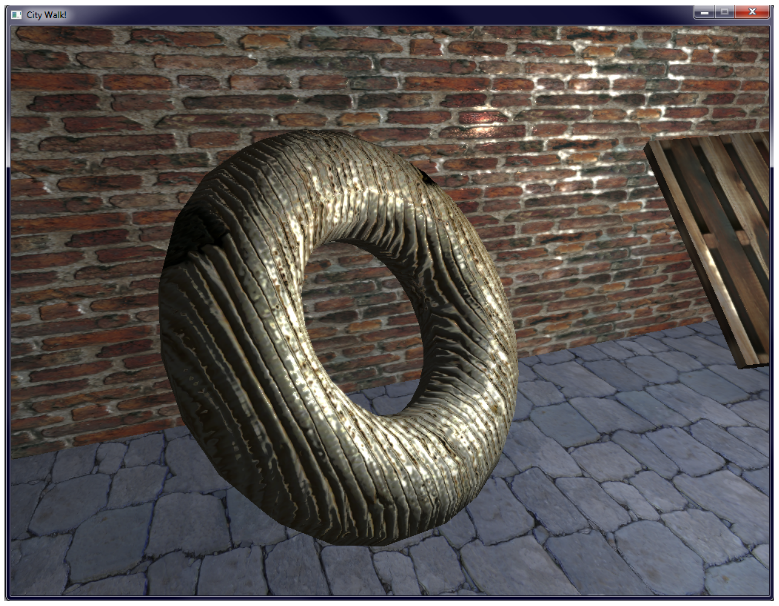

Figure 3: Bloom effect.

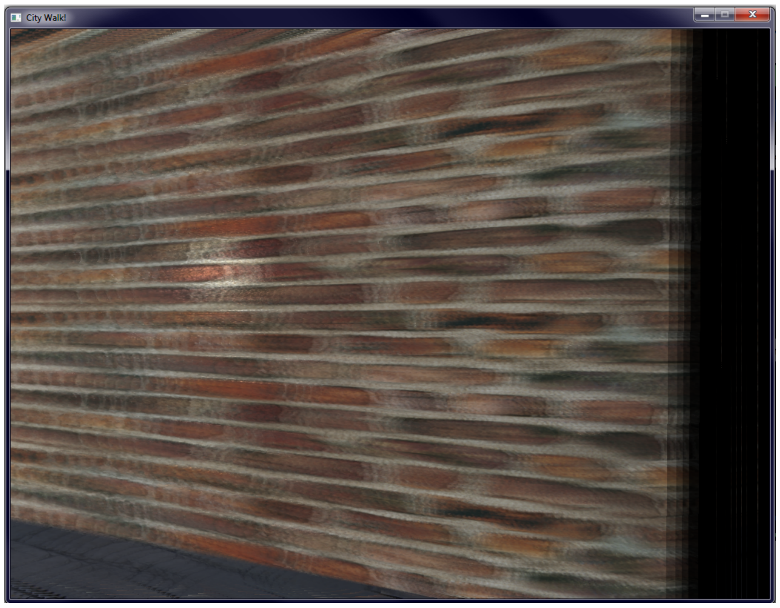

Figure 4: Motion blur effect.

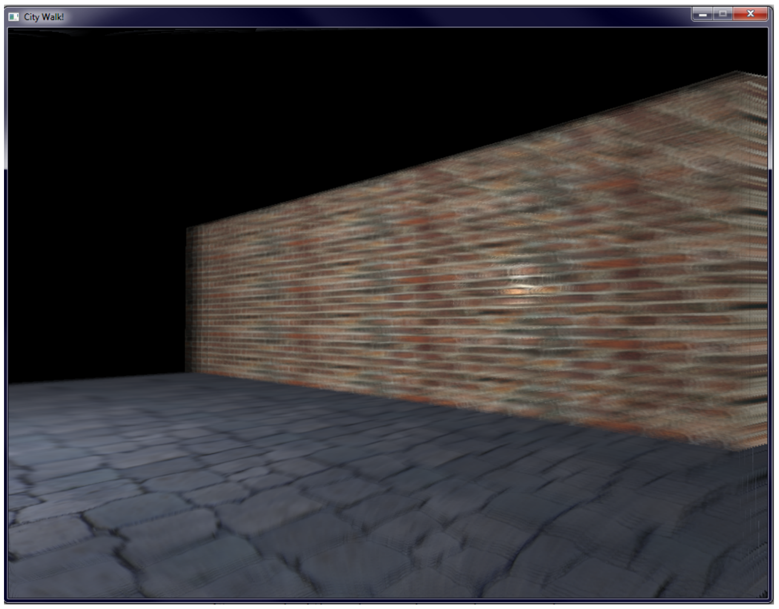

Figure 5: Motion blur effect.

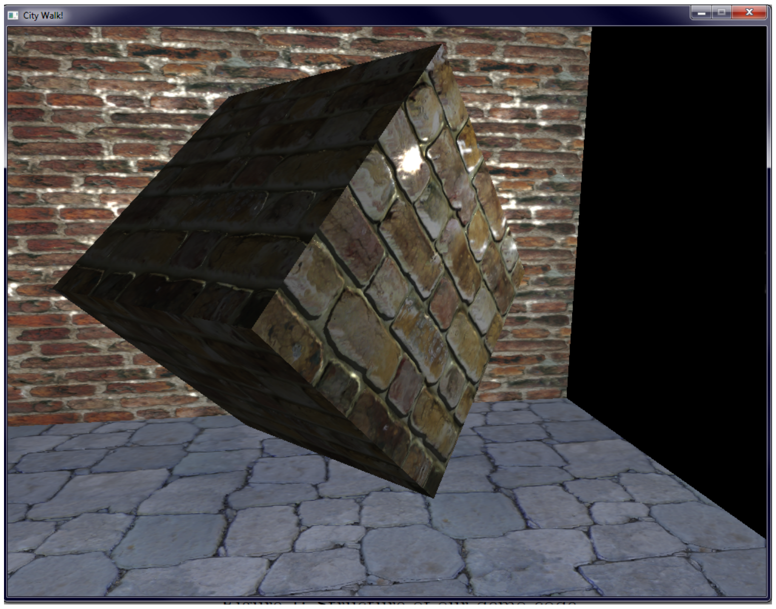

Figure 6: Parallax Mapping with Offset Limiting effect.

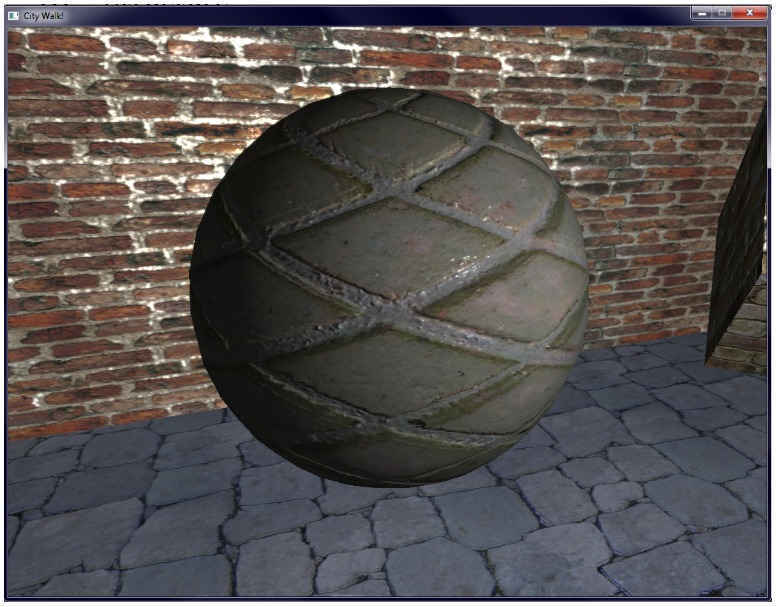

Figure 7: Parallax Mapping with Offset Limiting effect.

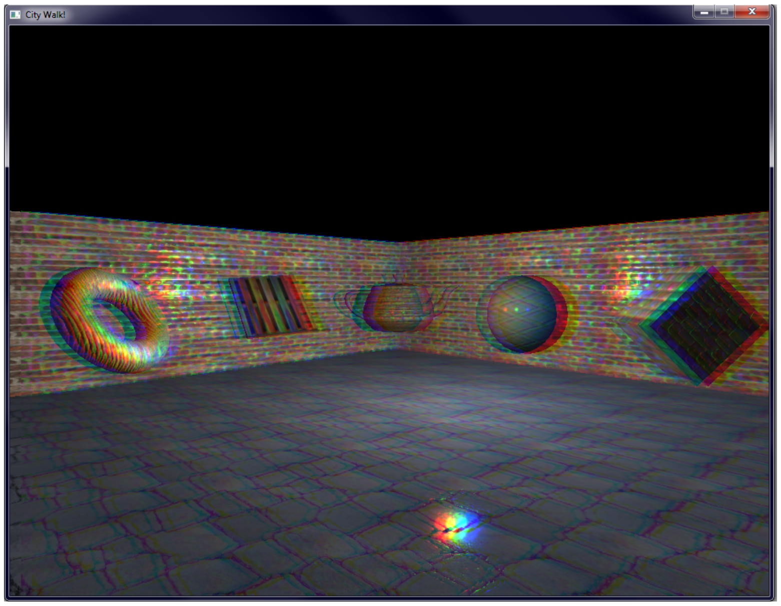

Figure 8: Scene Transition effect.

1.3 Problems

The main problems we ran into, were time issues from Stefan Doppelbauers side. As can be seen, the demo consists at the moment from one scene and 3(+1) effects, that were modeled and implemented by Martin Cerman. Stefan is currently working hard on his effects and his scene, and we expect the demo to be complete until the presentation.

A minor problem that needs to be mentioned as well is concerning the motion blur effect. When looking at the edges of the window, one can see artifacts created by this effect. Sadly we could not find the reason for these artifacts and therefore they could not be removed and will be considered as a feature (rather than a bug) of our demo (:-)).

References

[1] R. Fernando. *GPU Gems: Programming Techniques, Tips and Tricks for Real-Time Graphics*. Pearson Higher Education, 2004.

[2] H. Nguyen. *GPU Gems 3*. Addison-Wesley Professional, first edition, 2007.

[3] T. Welsh. Parallax mapping with offset limiting: A per-pixel approximation of uneven surfaces. Technical report, Infiscape Technical Report, 2004.