

Rendering Demo - Documentation

Story

The demo will take place within a single, large room. The idea is to keep the camera static initially and demonstrate the impact of various effect on the environment (perhaps with a variable degree of details). The room will be mostly lighted through a couple of point lights and Spot lights, these lights can also move.

Later, the camera will wander around the room and zoom in or rotate around existing objects to show the impact of various effects.

Another idea is to gradually add or remove effects during the demo to emphasize the impact of each separately.

Effects

The visual style of our demo will be comic or drawn like. To achieve this we use a hatching technique proposed by Bert Freudenberg in the Paper “Real-Time Halftoning: A Primitive For Non-Photorealistic Shading” [1], which was further explained in his thesis. There are also several other papers describing such techniques. We will also use indication maps, that show how visually important a portion of a model is. We will use the pipeline used in the paper “Real-time 3D rendering with hatching” [2]. This paper provides an illumination model and an automatic way to generate tonal art maps, which are the basic textures used for this style. To generate the tonal art maps following general steps are needed. First, characteristic information of a texture is extracted to produce a directional field. From this stroke trajectories are computed and from this a tonal art map is produced that satisfies some consistency constraints.

To further enhance the atmosphere we will use shadow maps for all important lights, moreover we plan to implement some post-processing effects, such as bloom, god-rays and contouring, as seen in the lecture. The contouring will be done by detecting edges in the normals and depth values in screen-space. In GPU Gems a simple post process effect for god rays is explained [3].

Implementation

For the hatching effect, we used a python program to generate the tonal art maps. This extracts with canny edge detection edge orientations. This is then interpolated for the whole texture using bottom up midpoint displacement[4]. Then using this direction field we generate strokes along these directions. One can set the number of control points and the displacement between control points.

The so generated textures are loaded into a texture array. In the shader we access the correct tone and fetch the corresponding texel. We used 32 tones per texture. In the shader the diffuse part of the phong illumination model is calculated. We then use this term multiplied by the

number of tones to get the texture corresponding to the tone of the fragment, because this correspond to the index of the tone in the texture array.

We also implemented shadows with depth maps. We did this for spotlights and pointlights. The depth value of a pixel from the point of view of the light is saved into a texture. This value is then used to check if a fragment is in light or shadow. The depth maps for the point lights are generated with a geometry shader that outputs the depth value from the point of view of the light into a cubemap. We use a geometry shader, because we can then write into all 6 faces of the cubemap in one pass.

The main pass of our pipeline renders the scene without effects, the scene with effects, the normal and the depth value into separate multisample textures.

Additionally we use transform feedback buffers for a particle system that is attached to spot lights and once active emits particles from the spot light's source.

Volumetric lights are made based on the VU slides along with many performance optimizations and tweaks added by us. We managed to get good performance on GPU without the need for downsampling.

In a next step we calculate contours by using a sobel filter and laplacian filter on the depth value and normal vectors of the scene. This detects discontinuities which correspond to places where a contour should be drawn. These are rapid changes in depth or normal.

The result of these passes are all written into different textures using different fbos of which the end result is blended in an additional shader.

We implemented sound with the irrklang library.

Unfortunately at the end, it appears that our demo is CPU bound and we are losing performance there. The issue was simply spotted once all the calculations for the demo were added in, which meant it is too late to add appropriate multi threading support.

Libraries

Opencv - texture generation

Assimp - model loading

Stb_image - texture loading

Irrklang - sound

Models

Models were taken from the internet from cgtrader. We found a nice asset pack there.¹

The models were then imported into blender and linked with the right textures where also the scene was modeled. The tonal art maps were generated with a self written python program.

¹ <https://www.cgtrader.com/items/636835/download-page>

Controls:

F2 - switch to debug cam

F6 - increase brightness

F7 - decrease brightness

Tested on Nvidia

Bibliography

[1] Bert Freudenberg, Maic Masuch, and Thomas Strothotte. 2002. Real-time halftoning: a primitive for non-photorealistic shading. In Proceedings of the 13th Eurographics workshop on Rendering (EGRW '02), Simon Gibson and Paul Debevec (Eds.). Eurographics Association, Aire-la-Ville, Switzerland, Switzerland, 227-232.

[2] Suarez, J., Belhadj, F., & Boyer, V. (2016, 04). Real-time 3D rendering with hatching. *The Visual Computer*, 33(10), 1319-1334. doi:10.1007/s00371-016-1222-3

[3] Kenny Mitchell. Volumetric Light Scattering as a Post-Process. GPU Gems 3. Chapter 13.

https://developer.nvidia.com/gpugems/GPUGems3/gpugems3_ch13.html

[4]Belhadj, F. (2007, October). Terrain modeling: a constrained fractal model. In *Proceedings of the 5th international conference on Computer graphics, virtual reality, visualisation and interaction in Africa* (pp. 197-204). ACM.