Philipp Koegel, Bastian Mattes, Jan Doerntlein *Erlangen, 07.01.2015*

GraPro: Graphics Project (Aufgabenblatt 5)

Aufgabe 1 [4 Punkte] OpenGL Framework. The following assignments are based on a OpenGL 4.4 engine that is written from scratch.

1. **Basic Setup (1 Punkt).** Wrapper classes for easy use of shaders, materials, debugging etc.
2. **Object loading (1 Punkt).** Use Assimp to load Wavefront obj/mtl files.
3. **Texturing (1 Punkt).** Implement Blinn/Phong shading and load a scene textured.
4. **GUI (1 Punkt).** Integrate a GUI library.

Abbildung 1: Possible result of assignment 1.

Aufgabe 2 [10 Punkte] Voxel Cone Tracing. The indirect illumination algorithms implemented in the previous assignments are computationally expensive. Using accelerating data structures like BVHs or kd-trees results in greats speed ups but still the time to render an image is far from real time. While this is almost impossible to accomplish with CPU rendering there are still ways to gain some more speed without a perceivable loss in quality.

The algorithm of Crassin, Cyril, et al. encapsulates a number of rays with approximately the same direction in a ray cluster. This cluster is represented as a cone. In a preprocessing step the scene is rendered from the light sources and the radiance and light direction is saved in an octree. The visibility and the energy of the cone can then be estimated very fast by testing against the voxels of the octree. If the result is not relevant for the upcoming computations all rays in one cluster can be discarded.

1. Octree (4 Punkte). Render from light sources and bake incoming radiance and light direction into the octree.
2. Filtering (3 Punkte). Filter irradiance values and light directions inside the octree.
3. Rendering (3 Punkte). Render from camera and sample BxDF using voxel based cone tracing.

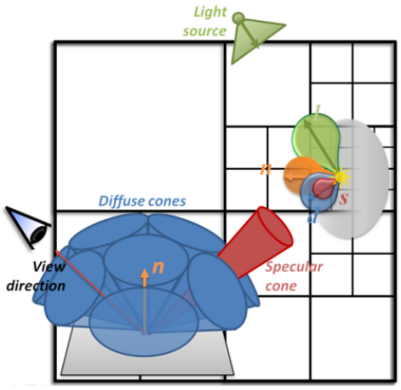

Abbildung 2: Voxel Cone Tracing.

Aufgabe 3 [4 Punkte] Extensions for assignment 2. To create a more realistic look of the voxel cone tracing algorithm it is extended with an ambient occlusion (not in screen space) technique here. Furthermore the meshes should support emissive material.

1. Ambient Occlusion (2 Punkte). Ambient Occlusion implementation.
2. Area Lights (2 Punkte). Emissive Material.

Abbildung 3: Possible result after assignment 3.

Aufgabe 4 [2 Punkte] Smoke Rendering Using Compensated Ray Marching. In this assignment the Aurora framework is extended to support smoke rendering. Due to the complex parts above this exercise is more like a bonus. The algorithm of Zhou, Kun, et al. is used in real time applications (on GPU) and promises fast results on the CPU as well. Several acceleration strategies and approximations are used in this technique.

The algorithm looks very sophisticated so maybe we cut it down to reduce the workload (and maybe the speed) while still implementing a simpler method to render smoke.

1. Density field approximation.
2. Resiudal field compression.
3. Light scattering.
4. Compensated ray marching algorithm.

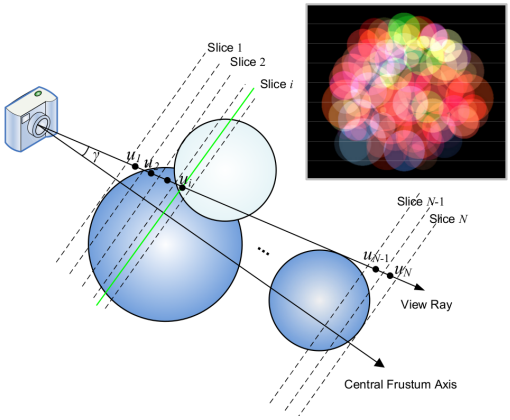

Abbildung 4: View pass in ray marching.

Sources:

1. Crassin, Cyril, et al. “Interactive indirect illumination using voxel cone tracing.” Computer Graphics Forum. Vol. 30. No. 7. Blackwell Publishing Ltd, 2011.
2. Zhou, Kun, et al. “Real-time smoke rendering using compensated ray marching.” ACM Transactions on Graphics (TOG). Vol. 27. No. 3. ACM, 2008.