Advanced Techniques

Francesco Andreussi

Bauhaus–Universität Weimar

10 January 2019

Bauhaus-Universität Weimar

Faculty of Media

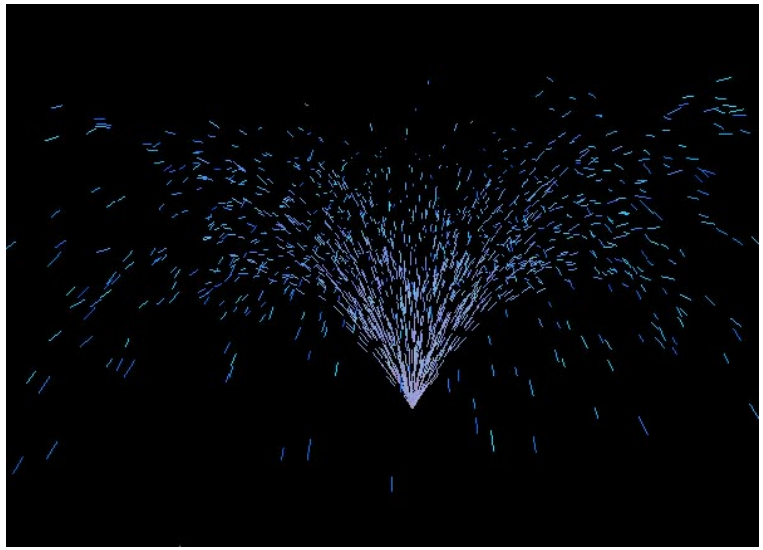
Particle Systems

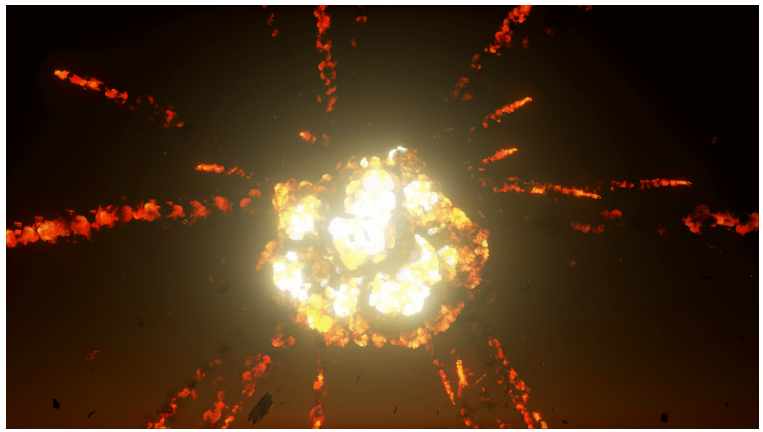
Particle Systems are huge sets of simple objects (points, camera facing quads, etc.) with similar randomised behaviour.

Every particle is spawned, has a life, dies and is re-spawned. During their lifetime, the attributes of the particles are updated. The usual attributes for a particle are position, velocity, color, remaining lifetime.

Particle Systems are efficient and effective for rendering amorphous objects (e.g. fire, smoke, snow/rain) and complex phenomena (e.g. explosions).

Tutorial1 Tutorial2





Visualisation (on globe)

It is possible to visualise geographically mapped data with OpenGL, using **Geometry Shaders**and **Transformation Feedback**.

Getting data from text or from a **displacement map**, i.e. a texture which can be used as source for calculating displacement for points generated by a Geometry Shader.

These data are updated every frame, allowing real-time modifications, and the Geometry Shader results have to stored in a buffer using the Transformation Feedback.

Geometry Shader Tutorial1 Tutorial2 Tutorial3
Transformation Feedback Tutorial
Earth Data Source

Text Rendering

There is no direct way of displaying text in OpenGL. The easier way of dealing with text is probably loading 2D or 3D letter models and arrange them in the virtual environment, but it is not a good way of doing it.

What it is actually done, is to render the characters as textures on some quads and there are two ways of doing it:

- **Bitmap fonts**: character are stored in one texture, the quads are calculated and the glyphs are retrieved manually.
- **TrueType fonts**: importing the fonts, the library automatically and dynamically computes the bitmaps and the quad sizes.

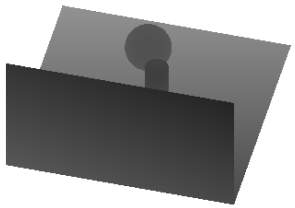
Bitmap Fonts Tutorial1 Tutorial2
TrueType Fonts Tutorial1 Tutorial2

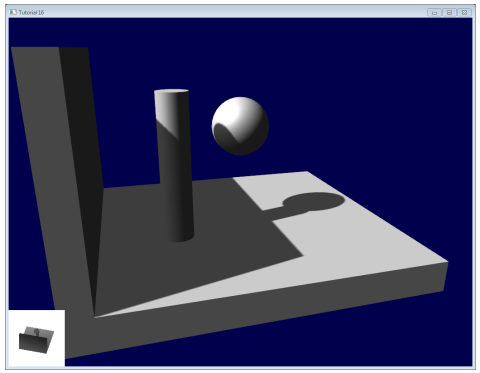
Shadow Mapping

Shadow mapping requires an offscreen rendering step for each light involved. In fact, the scene has to be rendered from the point of view of the camera, using the same shaders for every object and obtaining a depth texture.

Then, the scene is rendered normally, but, in the fragment shader, it is checked that the fragment is deeper (in light's projection space) w.r.t. the value stored in the depth texture. If this condition is true, the fragment does not receive light from that surface.

Shadow Mapping Tutorial1 Tutorial2 Tutorial3 Tutorial4 Tutorial5





Deferred Shading (1)

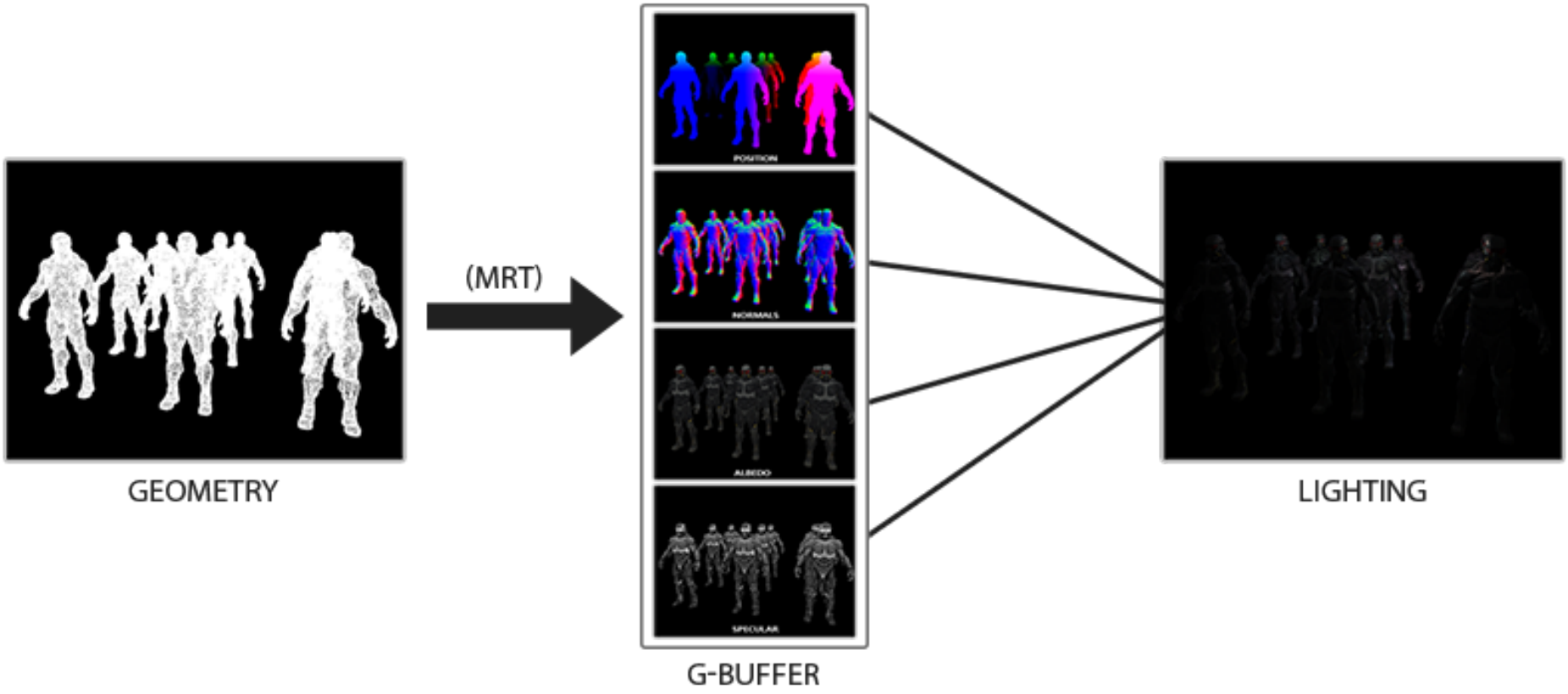
Interesting application of the offscreen rendering concept, it break down the rendering in two steps: getting all the geometrical information and using it to actually render.

The first pass renders the scene to the G–Buffer, where fragment positions, colours, normals and specular values are stored. The second step takes the textures stored in the G–Buffer and uses the data to compute the final appearance of the scene.

Deferred Shading improves performances computing the lighting only for visible pixels and makes the rendering of multiple lights more efficient. Nevertheless, it makes hard using transparencies and Anti–Aliasing.

Tutorial1 Tutorial2 Tutorial3 Tutorial4

Deferred Shading (2)



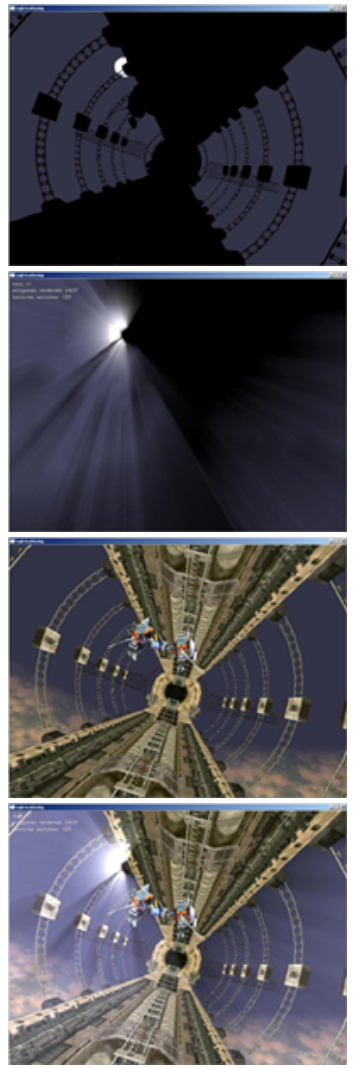
Light Scattering

Volumetric Light Scattering (aka God Rays) are a "special case" of Deferred Shading.

In fact the scene is computed multiple times: one for rendering the scene without the light scattering effect, and one for getting an image with the light source rendered.

The second image is modified applying light scattering formulas and, eventually, the two textures are blended together, and the result is displayed on the screen.

Tutorial1 Tutorial2



WebGL Port

WebGL is a slightly reduced version of OpenGL, designed to work on browsers using HTML5 and JavaScript.

High–level graphics framework (e.g. three.js) are available (but they should not be used).

Tutorial1 Tutorial2 Example

Thanks for the Attention!