ADVANCED GRAPHICS ENGINE BASED ON THE NEW OPENGL FEATURES

Juraj Vanek

Doctoral Degree Programme (1), FIT BUT

E-mail: xvanek29@stud.fit.vutbr.cz

Supervised by: Adam Herout

E-mail: herout@fit.vutbr.cz

Abstract: This work introduces an open-source OpenGL engine featuring many advanced effects used in realistic real-time rendering. Engine consists from scene manager which is based on material generator assigning dynamically generated material shaders to objects. Engine can be used in various applications requiring simple scene manager and which are aimed to generate high quality final image. Engine is easy to use and modular so there can be added many features in the future.

Keywords: OpenGL, engine, framework, real-time, realistic, render

1. INTRODUCTION

Today we witness rapid development in computer graphics which is closely related to development in graphics hardware. New graphics accelerators add many features allowing us to create such approaches/effects which was not possible with older hardware or was not running in real-time. However, direct programming under graphics interfaces like OpenGL and DirectX can be unproductive because of difficult implementation of some approaches. To ease this work, so called graphics engines are being created. Graphics engine is additional abstract layer above API functions allowing developers to use its advanced features with ease, for example, by using scripts and configuration files. It is vital for developers to have the ability to quickly implement new features.

Most of the graphics engines are proprietary, used in commercial applications like games and using DirectX API which runs only on operating systems from Microsoft. Although we have many free graphics engines running under OpenGL (which is free and multiplatform API), none of them is oriented to realistic real-time rendering or using newest OpenGL features.

My approach is an open-source OpenGL engine which implements most of the features from OpenGL 3.0+ and it is capable of rendering advanced effects used in real-time rendering like HDR lighting, ambient occlusion, dynamic shadows and other. Engine has been created in order to be easily updated in future and adding new features. It allows loading scenes with many objects and material generator which dynamically generates shaders for objects by setting only a few parameters. Thanks to OpenGL 3/4 support which is compatible with OpenGL ES engine works also on embedded devices (like mobile devices). Engine can be used in many applications requiring realistic rendering

2. EXISTING SOLUTIONS

In this section we will review some of the existing OpenGL engines. Although it is possible to use DirectX engines we will focus on OpenGL because it is an open standard which runs on wide variety of operating systems and devices and are used not only in gaming applications but also for technic and scientific visualizations. We can divide them into two groups:

Free – most of the OpenGL engines are free, either created and maintained by community (OpenSceneGraph) or it can be older gaming engine which has been proprietary but then released as open-source (id Tech 3). Examples:

- *OpenSceneGraph (OSG)*– widely used among developers in field of visualizations. It is based on scene graph implementation with many features to ease work of programmer. However, it is lacking support of newest features (currently only OpenGL 2.1 through extensions)
- *id Tech 3* – known as Quake3 engine, it was heavily used ten years ago in many games. Today it is obsolete and unsuitable for implementing advanced visual methods.

Commercial – are used in closed-source applications and games. Nowadays there are few gaming studios using OpenGL as their primary engine, but in professional applications like CAD systems are OpenGL engines dominant. Because such engines are not designated to realistic rendering, we are selecting two game engines:

- *Unigine* – nowadays the most advanced OpenGL engine with support of newest OpenGL 4.0 features (tessellation, instancing, advanced post-effects
- *id Tech 4* –is featuring many interesting features (Mega Texture – capable of working with extensively large textures used in terrain rendering, shaders, post-processing) but it is becoming outdated and will be replaced with id Tech 5

We can see from this brief overview that there is no free OpenGL engine focusing on advanced real-time effects except OSG. However, using such effects can be difficult as engine is not directly focused on them (still using old fixed pipeline with extensions instead of using fully programmable pipeline).

3. ENGINE FEATURES

Engine is implemented in C++ language and is using OpenGL 3.0+ core profile [1], so there is no fixed graphics pipeline and all vertex transformation and fragment processing is done entirely via programmable shader units [2]. Engine uses most of the features from OpenGL 3.3 API with some extensions from OpenGL 4.0. Currently, there are implemented these OpenGL 3.0+ features:

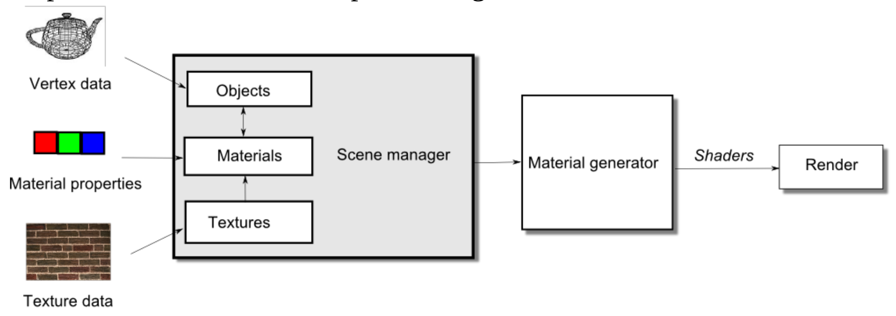
- No fixed-function processing
- Geometry instancing and geometry shaders
- Multiple render targets with multisampling
- Layered rendering into texture arrays/cube textures
- Shader variables binding through uniform buffers
- Shaders written in GLSL version 330+
- Texture gather functions for faster calculating of ambient occlusion

The core of engine is a dynamic material generator which generates shaders based on the material properties. These properties can be set manually in code or loaded from external file with 3D scene. Supported format is 3DS, textures are loaded together with objects.

3.1. DATA STRUCTURES

Main structure is scene manager which contains lists of all objects, materials, lights as well as all scene settings (for example resolution, viewing angle, antialiasing level etc.). To avoid loading objects and textures which have been already loaded, a texture and object cache is also present. Other variables are needed by camera rendering and to use with advanced rendering techniques, like framebuffer objects or render textures. Only one scene can be present at the time since it contains OpenGL context. Scene is composed from following structures:

- **Objects** – can be loaded from file, or created procedurally from basic built-in basic shapes like cube or sphere. Object has its own transformation matrix and pointer to the vertex buffer object stored on GPU for quick drawing. Object is linked with material through material ID.
- **Materials** – can be also loaded from file (custom shaders) or generated dynamically (as described in next section). Properties can be from basic material settings like color and surface texture to more advanced features like parallax mapping with multiple textures. Texture count is limited only by count of texture units available on GPU (currently, all GPUs supports at least 32 texture units)
- **Textures** – are loaded from external files in Targa format. I have chosen this format for its quick loading into GPU memory, however, disadvantage is size of those files on disk. Textures can be user-filtered from point nearest to anisotropic filtering
- **Lights** – holds information about light position, color and lighting model. Lighting model can be per-vertex (Gouraud) or per-pixel (Phong). Lights can also store framebuffer object in depth format for shadow map rendering



Picture 1: Engine pipeline

3.2. MATERIAL GENERATOR

Material generator is the main part, as it is capable of dynamically generate shaders based on material surface properties defined by user. These properties can be color or textures used to precisely define surface properties. Except multiple layers of color textures which can be combined either by adding, multiplying or blending, advanced surfaces can be defined. This includes bump, parallax, environment and cube mapping. Cube maps can be static or dynamic. Based on all these properties, material generator generates shader capable of execute commands leading to desired material surface. All shaders are generated in GLSL in version at least 330 [2], so graphics card compatible with this standard is necessary.

At first, vertex shader is being created. Default bindings with vertex attributes is set as well as necessary modelview and projection matrices. Depending on material type, there can be added texture samplers (for displacement mapping). If Gouraud light model has been selected, light calculations are done in vertex shader. Otherwise, it only sends and interpolates values (eye view vector and normal) to fragment shader. Shader is ended by vertex transformation using transformation matrices and vertex position (can be modified with displacement).

In fragment shader, one must activate textures associated with this material shader. Depending which textures are being used in material, appropriate texture samplers are added (for example, for color textures we have 2D texture sampler but for cube maps we have cube texture samplers). All texels from samplers are then combined according to mixing equations set by user. Final color is then sum of all textures and lighting model (if we are using Phong per-pixel light model). Fragment can be outputted to multiple render targets.

After shaders are linked and compiled, default vertex attributes are bound to materials, material settings are set (through uniform structures) and shader is ready to use. Other common variables (e.g. transformation matrix) are defined in uniform block and shared by all shaders.

3.3. POST PROCESSING AND OTHER EFFECTS

Thanks to modularity of the engine, a wide variety of real-time effects can be implemented. Most of those effects are dependent on rendering scene into the texture (serving as render target) which is further processed by shaders. Various advanced real-time effects can be used; some of them are following implementation from [3] and [5]:

- **Dynamic soft shadows** – using shadow maps with PCF filtering
- **HDR lighting** [4] – 16-bit or 32-bit render targets can be used, simple tone-mapping
- **Screen space ambient occlusion** – uses normal buffer, OpenGL 4.0 functions to speed-up

There is possible to use many more screen-space effects (like for example image processing) because it is possible to attach a custom shader to the material.

4. USAGE

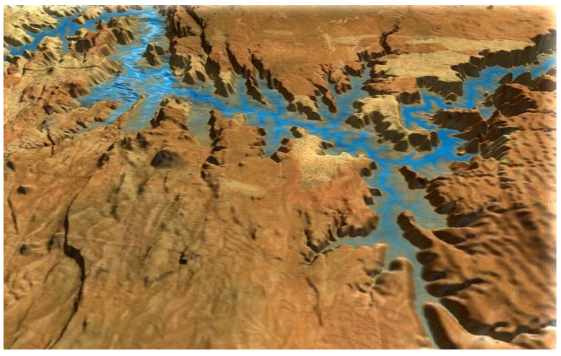
Usage of the engine is simple as it can be used in any C++ project by including main header file into project. Supporting external libraries (SDL, GLEW and lib3DS) are also necessary.

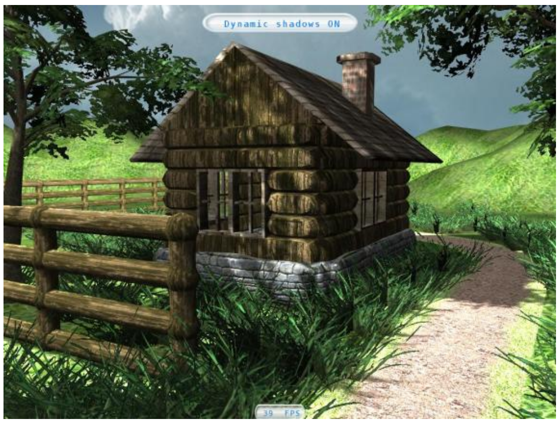
Scene initialization is done by creating `TScene` object and set its parameters. Most of these parameters are important to material generator which dynamically generates shader based on all of parameters. All those parameters can be set by scene initialization which runs exactly in following steps:

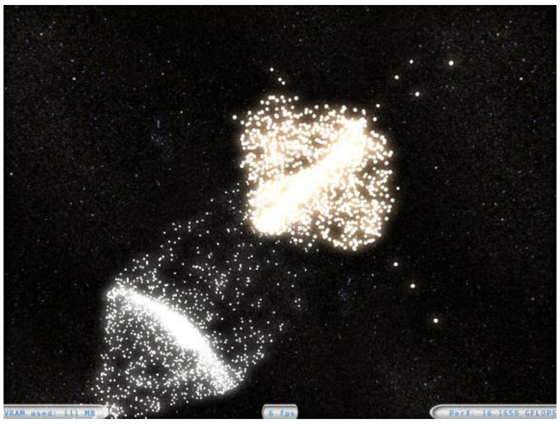
- Initialize scene, set viewing angle and near/far clipping planes
- Define camera according to type. Two camera types are supported – orbiting camera (user drags mouse and scene rotates around fixed point) or first person camera (inspired by 3D action games, simulates looking around with our eyes
- Add objects, either from built-in primitives or from external files
- Add materials, also with external textures and bind material to objects. Materials can be re-bound at any time or even replaced by other material
- Set special effects properties (e.g. exposure in HDR, shadow resolution in shadow maps etc.)
- Post-Init the scene where material generation will occur

So far, engine has been used to some of my projects starting from simple game through benchmark for graphics accelerators to a research application dealing with hydraulic erosion on GPU. Some of the sample renderings can be seen on following pictures:

VRAM used: 105 MB
45 fps







Previous screenshots illustrate various engine usages. Advanced effects demonstrated on the top left picture (HDR, ambient occlusion, shadows and normal mapping with hundreds of objects and shaders) runs on a low-end mobile graphics card (ATI Radeon 4650) at interactive frame rate 40fps in 720p resolution. On high-end cards (ATI Radeon 5970, dual GPU) frame rates are above 250fps.

5. CONCLUSION

Purpose of this OpenGL engine is there is no open-source solution existing which is aimed to realistic rendering in real-time. This OpenGL engine can be used (and was used) in various applications requiring very simple objects management and with high quality rendered image through using of advanced real-time rendering techniques. Thanks to OpenGL 4 and ES compatibility, engine is portable to various operating systems and architectures, from supercomputers to mobile devices.

Engine is well documented and easily to use because there are only a few commands required adding objects, materials, textures and link them together. Application can run in command line as well as part of some 3D canvas which can be used in GUI applications. There are existing applications based on this engine and one research paper application was launched on this engine.

Future work can be improving scene manager because currently there is no higher level object management like scene graph. Development could be much easier if there would be a GUI interface to edit scene (adding and editing objects and materials) and defining own scene description with some scripting language.

REFERENCES

[1] Segal, M., Akeley K.: *The OpenGL® Graphics System: A Specification (Version 4.0 (Core Profile))* [online]. 25.7.2010. URL: http://www.opengl.org/registry/doc/glspec41.core.20100725.pdf

[2] Kessenich, J.: *The OpenGL® Shading Language: Language Version: 400* [online]. 25.7.2010. URL: http://www.opengl.org/registry/doc/GLSLangSpec.4.00.8.clean.pdf

[3] Nguyen, H.: *GPU Gems 3*. 2007, 1008 s, ISBN-10: 0321515269

[4] Houlmann, F., Metz, S. : *High Dynamic Range Rendering in OpenGL* [online] 11.9.2006 URL: http://transporter-game.googlecode.com/files/HDRRenderingInOpenGL.pdf

[5] Mittring, M.: *Finding Next Gen – CryEngine2*, SIGGRAPH 2007 [online]. 2007. URL:http://ati.amd.com/developer/SIGGRAPH07/Chapter8-Mittring-Finding_NextGen_CryEngine2.pdf