University of Borås
Department of Information Technology
Course: Computer Graphics
VT 2017

Assignment 3: Materials, Shading and HLSL

Objectives

Designing a basic render system for 3d model components including custom effects based on your entity system component based game engine and creating a rendering effect, supporting fog, bump mapping and environment mapping in HLSL. The program, implemented in MonoGame, should show a virtual 3d environment with consistent lighting and a set of objects.

Task

Your program should create a scene with some 3d objects and use material settings, lighting and effects to make it look realistic (and nice).

Requirements

- **Function**
    - The program should use several objects loaded from 3d model files. An archive with some 3d objects and textures is available in PingPong (the models are licensed under GPLv2). You may use these but are not required to do so. You may also use generated objects in the scene in addition to some models (e.g. if you already have code for procedurally generated ground).
    - All objects in the scene should be shaded consistently using the same light source(s). At least ambient light and one directional light source should be used. It should be possible to change the lighting of the scene globally, potentially at runtime.
    - The program should render the scene with distance blending or fog to hide the borders of the limited world. The fog or distance blend should be applied consistently to all objects.
    - The scene should contain at least one object rendered with your own shader effect, either
        - one combining environment mapping with bump mapping to produce a material that is both lit according to the Blinn-Phong model and reflects the surrounding environment; or
        - one that make objects receive and throw shadows on both itself and other objects in the scene (Shadow Mapping).

      Choose **one** shader effect that seem to be the most relevant for your game in the Game Programming course.
    - Your shader effect combining environment mapping with bump mapping should meet the following requirements:
        - The shader effect should use an environment cube map dynamically generated from the scene (at startup or continuously).
        - The shader effect should use a normal map texture that contains normals in the normal-tangent-binormal-frame. This is the most common type of normal map.
        - The shader effect should have material parameters to set the Blinn-Phong material properties Diffuse color, Specular color, Specular power (Shininess), Normal texture, Environment map texture and a factor to scale the contribution from the environment map from 0 to 1 from the application.

        - The shader effect should support colored ambient light and at least one colored directional light source (with separate parameters for diffuse and specular light intensity) settable from the application.
        - The shader effect should support fog consistent with the rest of your scene.
    - Your shader effect that make objects receive and throw shadows should meet the following requirements:
        - The shader effect should use one shadow map per light, dynamically generated from the scene (at startup or continuously).
        - The shader effect should use a blur-kernel to make the shadow have a smooth appearance.
        - The shader effect should have parameters to set the Blinn-Phong material properties Diffuse color, Specular color, Specular power (Shininess) and a factor to scale the strength of the shadows from 0 to 1.
        - The shader effect should support at least two shadow throwing colored lights of some sort (directional, spotlight, point), with separate parameters for diffuse and specular light intensity.
    - It should be possible to freely move the camera around and rotate it using keyboard and/or mouse so that the scene can be viewed from any position and direction.

- **Structure**
    - Your program should follow entity system component principles in its design. Aim to build a framework for your entity system component based game engine that handles placement and rendering of 3d models. Make the main program a client that uses your game engine to show a scene.

Hints

Remember to configure the content importer to generate MIP maps for the textures to reduce the performance impact of large textures.

To render a mesh double-sided, you can render it with cull mode set to none in the render state. This leads to inconsistent shading of the back face (since it will be shaded using the normal for the front face). The shading can be made consistent for the reverse side by using a custom pixel shader that flips the normal for pixels that are viewed from behind the surface (the sign of the scalar product between the normal and the direction towards the viewer may be useful here). Alternatively, the mesh could be drawn twice with back- and front-culling, respectively. The back face would still need a custom shader that considers the direction of the normal.

When importing a model each ModelMeshPart in the model will contain a BasicEffect instance (or some other effect if you set that in the content importer) set up with the material properties specified for this part in the model file. Copy these properties when/if you replace the BasicEffect with some other effect (assuming the new effect needs those material properties) to preserve the material of the part.

Environment reflection mapping is an approximation – don't expect the reflection to be exactly right and don't place the reflecting object too close to other objects. The camera view and up directions needed to generate each face of the cube map are not entirely intuitive. Generating and testing just one face at the time is a good way to check the orientation. There is also a table of camera directions in Lecture 8.

If you are using tangent vectors generated by the content pipeline do double check that they come out alright, there have been problems for some objects. Also check that your normal maps look blue – otherwise you will need to adapt the order of the x, y and z components in the resulting normal compared to the examples in Lecture 8.

Extras (voluntary)

- Make the mirror object seem translucent too by adding refraction to the environment reflection mapping effect.
- Animate some object(s). Note that the objects imported from FlightGear has a rather bad internal structure – usually all parts have the same origin so you need to find appropriate centers for any rotations manually.
- Add a skybox to simulate the distant surrounding environment. With a careful selection of fog color, distance blending should still look good.
- Add some post-processing effect(s).
- Optimize the mesh drawing further by adding culling at the application level so that only meshes that could possibly be in view are sent to the graphics hardware. Each ModelMesh has a bounding sphere that can be used for this purpose.
- Implement good best effort rendering of translucent objects, i.e. so translucent objects do not block other objects completely and the ordering between opaque and translucent objects is correct. This may also include support for rendering some meshes as double-sided, where appropriate due to the model, e.g. windows and any opposite "wall" to prevent it from disappearing. You may assume that the user notifies the framework about meshes that are translucent and/or double sided, but provide the means to do this notification. The Nordstern Zeppelin model, which contains many translucent meshes, and the snow plow, which contains some, can be useful for testing these aspects of your render and scene management.

Formalities

Work in your existing group. You may build on your previous work for assignment 1 and 2. A zip-file containing the full solution with all source code and an executable file should be uploaded to Ping Pong by the deadline set for the assignment (see KronoX and PingPong).