2018-12-20

BTH

3D Programming – DV1568/DV1542

Project – 8 hp

1. Description

In this project you will implement an Interactive 3D Application.

1.1 Rules:

- All GPU code (Shaders) must be 100% of your authorship, and you should be able to explain everything inside them.
- You can reuse other libraries, such as math, sound or window management (DirectXTK, ASSIMP, SDL, SFML, GLFW, etc)
- The CPU side of the application must be written using C/C++
- Debug layers MUST be activated in DirectX and OpenGL to catch ALL errors with the APIs.
- The APIs you can use are OpenGL 4.3+ and Direct3D 11
- You are encouraged to work in groups, up to 3 participants.
    - All members in a group should understand all the code submitted.

1.2 Mandatory techniques or implementation details:

- You have to use a “perspective” camera,
- You have to use textures in your 3D models,
- You have to implement ambient, diffuse AND specular phong shading per pixel.
- You have to implement one camera control mode: either FPS or ORBIT,
    - FPS: Use “w-a-s-d” for moving and the mouse for looking at different directions
    - ORBIT: Use “w-a” for zoom in/out, mouse to orbit around point of interest.

- Using a **Geometry Shader** is not mandatory, but recommended.
- At the end, you will submit the project and also a written report in English describing each of the techniques you have implemented. The English language is not graded in any way, so do not worry about it, but use a spellchecker before submitting the report. In your report, explain briefly the main aspects of your technique. For instance, do not explain "how deferred rendering works in general", but "what are the details of my implementation of deferred rendering".
- It is ONE report for the entire group.

1.3 Additional Techniques

Additionally, to the mandatory techniques described above, you will implement a number of techniques from each category in the list below. The number in **RED** inside the parenthesis states how many from each category have to be implemented if you are in the course DV1542 or if you a game programmer but in the course DV1568. If you are a Technical Artist in DV1568, the number in **BLUE** inside the parenthesis states how many from each category have to be implemented.
For example, for the "Core Techniques" you have to implement **ONE (1)** of the options.

- Core techniques (**1**,**1**):
  - **Deferred Rendering**
  - **Skeletal Animation** (Can use library to load animations)
- Geometry (**2**,**1**):
  - **Parsing and rendering an existing model format** (OBJ, MD5, etc). This includes parsing and using the material defined in the format (textures). If you use a library like Assimp, it does not count.
  - **Displacement mapping using Tessellation hardware**
  - **Height-map terrain rendering, user can walk on the terrain using WASD keys.**
  - **Level of Detail using Tessellation Hardware**
  - **Morph-based vertex animation**

- Texturing and Lighting (**1**,**2**):
  - **Normal mapping**
  - **Screen Space Ambient Occlusion**
  - **Blend mapping (multi-texturing) – see slides**
- Projection Techniques (**1**,**1**):
  - **Shadow mapping**
  - **Dynamic cubic environment mapping**
  - **Dynamic paraboloid mapping**
- Acceleration Techniques (**2**,**1**):
  - **View frustum culling against a quadtree**
  - **Portal Culling**
  - **Front to Back rendering**
  - **Back face culling using Geometry Shader**

- Other techniques (**2**,**3**):
  - **Particle systems with Billboarded particles**
    - **If Particles are updated on the GPU, counts as 2**
  - **Gaussian filter in Compute Shader**
  - **Bilateral filter in CS (any filter that can be applied in two passes)**
  - **Mouse Picking**
  - **Screen-Space Antialiasing**
  - **Water-effect (based on vertex or texture animations)**
  - **Glow-effect**
  - **Make a small game in your application**

- If you want to propose other techniques, please contact fll@bth.se with ideas before starting any implementation.

1.4 Guidelines:

We do not assess on how nice the models or the textures look, but on the correctness of the techniques and your understanding. As explained in the lectures, many techniques have variations, and it is ok to implement a variation as long as you understand the main idea.

It should be clear when presenting the project, who did what in each group. The project presentations are **individual**, and do not affect the other members of the group. ALL participants in a group should know about ALL techniques in the project, and can be questioned during the presentations.

If you have worked on Linux or Mac you will have to bring your computer to present your project, it is not enough with showing the code, the application has to be showcased in front of the teacher/s.

FILES and other information will be uploaded to Canvas to support the implementation of some of the techniques (textures, models, animations, etc)

2. General Considerations

The following is a guide about the project presentation and assessment.

- The program should not crash. Not even at the end of its execution.
- The program should not contain major memory leaks (visible memory consumption increase in the task manager)
- The usage of the API should be "correct". For example, it is not ok to re-create a Vertex Buffer every single frame unless there is a clear reason or need for this.
- If the implementation of some of the techniques is not correct, then we will make a note on that technique, and continue the presentation. The project will not be passed until this is fixed. This is not the same as to submitting your project incomplete (without the techniques), which is not acceptable.
- **Most variations of behavior in your code should be possible to do without closing the application (and re-compiling). IF you find yourselves changing the code and recompiling to showcase please consult us before submitting.**

2.1 Specific comments about some of the techniques

Core techniques

- Deferred rendering and lighting

- Basic deferred rendering, with at least one light pass at the end rendering a full-screen quad or a full screen triangle.
- Basic Phong shading with a point light (ambient, diffuse, specular)
- **Should be easy to verify that removing each term of the lighting equation produces the expected results..**

- Skeletal animation (m3d, FBX, Collada, glTF, etc.)

- get the animation to play, and correctly interpolate the poses.
- if you load an m3d for animation (from Frank Luna's book), this would **also** count as "Parsing and rendering of an existing model format"
- You can use a library to load the animation data. Eg. Assimp or others. More information will be provided when we cover this in the lectures.

Geometry

- **Parsing and rendering of an existing model format (OBJ, MD5, etc.).**

    - should support at least **one** basic material with at least **one** texture from the material. The material has to be loaded from the mesh or material file, **not hardcoded** in the code.
    - the mesh does not necessarily have to be "in parts" (no need for multiple materials for different parts of the mesh)

- **Tessellation with displacement mapping.**

    - This should be a coarse mesh, that gets tessellation applied in the GPU pipeline and then displaced using a displacement map.

- The tessellation level can be fixed (hard-coded)
- To show this, you need to be able to disable the tessellation and the displacement.

- **Level-of-detail using tessellation**

  - can be terrain, or a basic model.
  - the criteria for simplifying the mesh can be camera distance or controlled by the keyboard.
  - if it is a terrain, does not have to be "perfect" alignment between patches.

- **Height-map terrain rendering, user can walk on the terrain.**

  - the "walking" on the terrain feature **does not have** to be smooth.
  - the terrain map can be fully regular (same size of triangles everywhere)
  - the number of vertices horizontally and vertically can be the same as the texture, but be careful with the size of the texture…
  - You can also use a tile-able texture, and repeat the displacements in the terrain.

- **Morph based vertex animation**

  - Any form of Morphing between keyframes for a set of vertices.
  - The key part is having a set of keyframes, along a time line, and being able to specify a time between 2 keyframes and get the vertices linearly interpolated. This can be done in the CPU every frame, or store the keyframes in a buffer in the GPU and interpolate in the vertex or geometry shader (preferred).

Texturing and illumination

- **Normal mapping**

  - you can reuse existing maps from other sources. We will upload some models.
  - **You need to understand how the TBN** (tangent,bi-tangent,normal) matrix is built and used to adjust the normal when presenting this technique.

- **Blend-mapping**

    - show the blending working correctly, the render state settings, blending equations and the Shaders sampling the textures.

Projection Techniques

- **Shadow mapping POPULAR TECHNIQUE…BE PREPARED FOR THESE QUESTIONS:**

    - do you know how to change the shadow map resolution?
    - do you use PCF?, if yes, do you understand the basic idea?
    - Dooes your technique suffer from self-shadowing or acne, do you understand the reason?
    - do you understand why is the "perspective divide" applied to the position projected with the Light WVP?
    - do you know how is the transformation from projected position pos to (u,v) space in the shadow map?

- **Dynamic cubic environment mapping**

    - reflections have to be updated dynamically as the camera moves on the objects that have this feature.

- **Dynamic paraboloid mapping**

    - reflections have to be updated dynamically as the camera moves on the objects that have this feature.

Acceleration techniques

- **View frustum culling against a quadtree**

    - you can build the Quadtree when the application starts
    - ideally you should work on bounding volumes (spheres or AABB) surrounding your objects
    - the depth of the quad-tree has to have at least 4 levels, but allow to define a different number! (use recursion)

- it can be sparse or regular (a sparse would be a quadtree that has not been further subdivided in certain branches because there is no geometry there)
- You have to show that the culling is happening outside the frustum.
    - This can be shown in several ways, but the best way is to have a separate camera, looking from the top downwards at the scene, and be able to move the camera used for culling with the keys to see the geometry being culled.

- Back-face culling using geometry shader

- Need to prove that works correctly! not just eye-balling the code.
- For example, can make a triangle to rotate and face back towards the camera.
- Need to show that culling is disabled in the fixed pipeline.

Other techniques:

- **Particle system with Billboarded particles**

    - Particles update can be in the CPU,
    - unless the particles are always at the same height of the camera, they should be aligned to the camera looking direction, and not just one up axis (Y)

- **Picking**

    - some feedback that the picking works. Picking against any shape will do it, but it has to work properly not just in the center of a big sphere.

- **Screen-space anti-aliasing**

    - We only cover FXAA, but in any case you should have a basic understanding of how the technique works.

- Water-effect

    - Water movement, there are many ways to achieve some water-like movement.

    - We do not care how it looks in terms of refraction and caustics, or collisions with other objects in the water.

- Glow-effect
    - should be able to adjust the intensity of the glow (it is possible to recompile to change)

- Make a small game of it
    - The game should be minimal, but playable. You can re-run the application to restart the game. The game should have some basic goal, and the user should be able to affect in some way the surroundings (picking up objects, shooting and destroying targets, etc).

3. Submission, deadline and presentation

You should submit your solution to Canvas.

The report should be submitted along with the solution. If there is no report the solution will not be graded.

Before submitting, please “clean the solution (both Debug and Release modes), and delete the files with extension “.sdf” from the project folder. After this, make a ZIP file of the project folder with the report.

IMPORTANT:

The assessment of your code is very important, please do not take this lightly. You should be able to explain your code (not the code provided), why it works, and what happens if you make changes to it. You should be able to explain the math that you use in your code as well.