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

General considerations:

- 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 (for example, resizing a VB is not possible in DX11)
- If the implementation of some of the techniques is not correct, then we will make a note on that technique, and continue the interview. The project will not be passed until this is fixed.
- If a student fails to explain the code, we will make a note, and at the end of the day we will inform the student that he will have to present the project again.
- You **can** restart the application to activate or deactivate some feature.

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 fullscreen quad.
- Basic phong shading with a point light (ambient, diffuse, specular)
- Try removing each term of the lighting equation and see that the results still make sense.

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

- get the animation to play, and correctly interpolate the poses.
- if you load an m3d for animation, this would also count as "Parsing and rendering of an existing model format"

Geometry

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

    - should support at least one basic material with at least one texture. A material can be simply a colour. The material has to come from the mesh or material file, not hardcoded in the code.
    - the mesh does not has to be "in parts"

- **Tessellation with displacement mapping.**

    - This should be a coarse mesh, that gets tessellation applied in the GPU pipeline and then displaced using a normal map.
    - The tessellation level can be fixed (hard-coded)
    - You need to be able to switch between the coarse and the refined mesh

- **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)

- **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.

Texturering and illumination

- **Normal mapping**

    - you can reuse existing maps from other sources.
    - You need to understand how the TBN (tangent,bi-tangent,normal) matrix is built and used to adjust the normals.

- **Blend-mapping**

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

Projection Techniques

- **Shadow mapping**

  - do you know how to change the shadow map resolution ?
  - do you use PCF ?, if yes, do you understand the basic idea ?
  - do you 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) in the shadow map.

- **Shadow volumes**

  - compute the shadow volumes, can be in CPU or GPU
  - understand the algorithm presented in the lectures

- **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 (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 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.

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.

- **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 covered 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)

- Solve any task using compute shaders
    - combination of Compute Shaders and any of the previous listed tasks will give you 2 points. Eg. blurring using CS will give you 2 points in this section.
    - **if you find another task not listed above then using Compute Shaders will count only as 1 point (ask the teacher before implementing)**

- 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.

Thanks!

Francisco