Computer Graphics exam project descriptions

Version 1.0, 15th May 2023.

This year, you will have to define yourself the topic of your project. It can be anything using 3D or 2D visualization created using Vulkan. It can be for example:

- A simple video-game
- A product showcase video
- A didactic tutorial on some subject
- A simple CAD, to project rooms, recipes, or other entities
- A 3D visualization of a data-set
- … or any other application which can benefit from a 3D visualization, or an advanced 2D view which exploits a z coordinate to create different overlapping layers.

However, you will have to define the topic of your project at the beginning, and fill it in a form (as it will be detailed later).

Projects must be done in teams of two students. Under motivated circumstances, party of one or of three students will be allowed: the expected distribution of number of team members is: 10% single person, 60% two people groups, 30% three people groups – i.e. in a group of 200 students, 20 are expected to do the project individually, 120 in 60 groups of two, and 60 in 20 groups of three. All team members will have to discuss the project in the **same** exam date. Remember however that this year, assignments will be evaluated off-line: if someone cannot prepare the assignments, might come for the project discussion, and she will complete the exam in a future session. *Exception will be possible only for **VERY MOTIVATED AND SPECIAL** circumstances*.

A link to a Microsoft Form will be provided to register for the project: there you will be able also to specify your curriculum, your other team members, and define the *topic of your project* (i.e. what you will be presenting at the exam). *The **link** will be available **in the WeBeep page of the course***. Please keep in mind that for group projects, each member will have to register individually using the Google form: this helps confirming that every participant of the group is indeed interested in doing the project with the other members of the party. *Students that have not registered the project on-line*, **cannot be admitted to discuss it**.

Registration for the projects will be open until mid-January 2024: there is no deadline, but must be done before the presentation. The suggestion is to register the project when it is at about 20% of its implementation stage – I.e. when both the group members and topics are final. Remember that for groups of one or three students, your registration can be denied due to the lack of motivation for the non-conventional size of the party – this might creating problems when registering too late. Strange topics, considered not relevant for the course, might be rejected as well. Registering too early, instead, can create problems due to possible topic or group members changes during the implementation of the idea.

Each project must be done in C++ using Vulkan. Different requests can be discussed (i.e. using Direct X 12, or a Vulkan to Python Bridge), but must be explicitly allowed before the exam. You are allowed to start from some of the example code and the assignments given during the course. You can even start from other public sources available on the web: however, in this case, you will be required to know exactly what every line of that code does.

You are allowed to use third party libraries and helpers such as *Vulkan.hpp (https://github.com/KhronosGroup/Vulkan-Hpp/blob/master/vulkan/vulkan.hpp)*, but ***not*** *complete 3D engines*: if in doubt, please ask before basing your code on some material found on-line.

Projects will have to showcase what you have learnt during the course. Your code will need to have sections where: it explicitly loads the geometries and the textures, sets the shaders (which must be written by yourself), sets the vertex formats and the uniforms, creates the graphics pipeline, and records the draw-calls in the command buffers to finally show the scene on screen.
Your application will show objects that are defined through a set of local coordinates, which are mapped to screen after a set of transformations. In particular, they must undergo first a local transform to place them in a given position in a virtual world. Then they must be subject to a view transform, that will select which part of the virtual world should be visible, and finally a projection that will convert the selected area of the world to normalized screen coordinates, ready to be transformed into screen coordinates by the drivers.
World matrices should be composed combining translations, rotations and scaling. Rotations can be expressed either using Euler angles or quaternions. View matrices can be computed either with the look-at or with the look-in-direction techniques. Projections can be either parallel or perspective.
Objects must be encoded as meshes, whose vertices are characterized by several parameters. These parameters should include the position, but also the direction of the normal vector to the surface, the texture coordinates, and maybe the tangent or a color. Meshes should be either loaded from an external file, using formats such as OBJ or GLTF, or dynamically created with an algorithm. The corresponding vertex formats should be defined: in particular, an application can use several different formats at the same time, each one aiming at reducing the memory occupation, by offering only the attributes really needed for a specific mesh, rendered on some particular conditions.
Navigation controls should be added, to allow the elements of the scene to be viewed either in first or third person. Navigation can occur using keyboard, mouse input, joystick or a mixture of different controls. The user should be able to interact with scene elements, that will respond to their input – for example, moving the camera or some of the objects or lights in the scene.
The final color of the pixels on screen should be determined using shaders. In particular, one or more pipelines should be defined to approximate the rendering equation in different ways. Each pipeline will support its own set of shaders, and it will start from a given vertex format.
Shaders will communicate with the application using uniform buffers, where they will receive, for example:

- The world, view, and projection matrices (either separate or as a single entity).
- The position, direction, color and parameters for the lights.
- The textures required by the shaders.
- Other parameters necessary to compute the BRDF, such as the exponent in Phong or Blinn specular model, or a threshold in a Toon shader.
- The position of the viewer.
- And whatever necessary to achieve controllable effects.

Several data sets layout could be available, and different level of bindings could be used to reduce the communication overhead between the CPU and the GPU. Vertex shaders will compute the position of the vertices in normalized screen coordinates, they will pass other parameters to the

fragment shaders, such as texture coordinates and transformed normal vector directions, as well as other parameters, such as a light direction computed per vertex or a color.
The final color of the pixels will be computed with fragment shaders, that will implement a model such as the Lambert or Oren-Nayar diffuse, the Toon shader, or the Blinn, Phong or Ward specular models, or the Cook-Torrance, using either GGX or other distribution or geometric term functions. They can use both direct lighting techniques, and indirect light approximations, such as ambient or Image Base Lighting. Direct lighting can include directional sources, point sources or spot-lights. The parameters of the material, such as diffuse and specular colors, emission, roughness, metalness, could be either constant or read from textures. Textures can also be used in advanced rendering techniques such as normal maps or detail textures.
The final quality of the images generated will also be considered: after all, it is a computer graphics course, and the final appearance will also be evaluated. Although the most important thing will be the techniques being used, the overall size of the virtual world will be considered, as well as the introduction of extra elements, such as animation, basic physics simulation, intelligence in some algorithms, on-screen display and scene change, and so on.

Each project will require a set of 3D assets to be completed. An extensive set of assets will be provided, with details given on the WeBeep page of the course. They will include food, furniture, cars, buildings, vegetation (both modern and in medieval settings), simple daily life objects, dungeons, terrain, schools, hospital wards, and many other. Textures and PBR materials, as well a HDRI backgrounds will also be also provided. However, the use of different assets either done or obtained by the students, is encouraged and will be positively evaluated during the presentation.