Practical Assignment CG - 2020/21

MIEI & LCC DI - UM

The goal of this assignment is to develop a mini scene graph based 3D engine and provide usage examples that show its potential. The assignment is split in four phases, each with a due date as specified in the course page.

For each phase a set of configuration XML files will be provided for testing and evaluation purposes. Each configuration is accompanied by the respective visual output. Students should verify that their output is identical to the sample images provided for each configuration file.

Phase 1 – Graphical primitives

This phase requires two applications: one to generate files with the models information (in this phase only generate the vertices for the model) and the engine itself which will read a configuration file, written in XML, and display the models.

Generator

To create the model files an application (independent from the engine) will receive as parameters the graphical primitive's type, other parameters required for the model creation, and the destination file where the vertices will be stored.

In this phase the following graphical primitives are required:

- Plane (a square in the XZ plane, centred in the origin, subdivided in both X and Z directions)
- Box (requires dimension, and the number of divisions per edge)
- Sphere (requires radius, slices and stacks)
- Cone (requires bottom radius, height, slices and stacks)

For instance, if we wanted to create a sphere with radius 1, 10 slices, 10 stacks, and store the resulting vertices in a file named sphere.3d, and assuming our application is called *generator*, we could write on a terminal:

```
C:\>generator sphere 1 10 10 sphere.3d
```

Instruction to generate a box with 2 units, where each side is divided in a grid 3x3:

```
C:\>generator box 2 3 box.3d
```

To generate a cone with radius 1, height 2, 4 slices and 3 stacks we could write:

```
C:\>generator cone 1 2 4 3 cone.3d
```

Finally, here is an example to create a plane with 1 unit in length, and 3 divisions along each axis:

```
C:\>generator plane 1 3 plane.3d
```

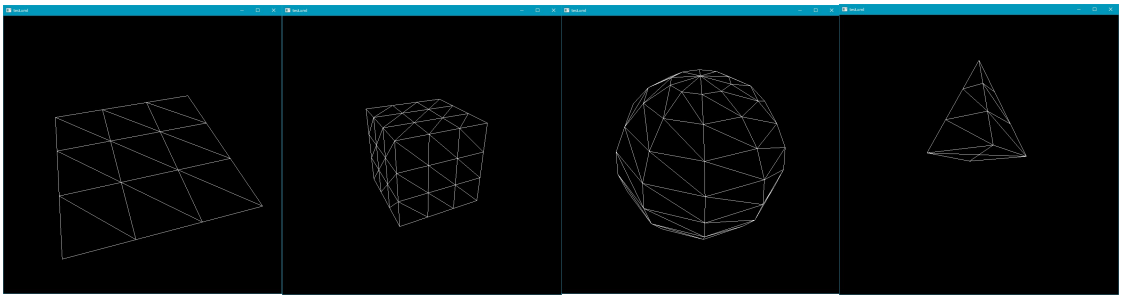
After the above commands are complete there should be a set of files with "3d" extension, with all the vertices and faces required to define the required primitives.

The file format should be defined by the students and can support additional information to assist the reading process, for example, the file may contain a line at the beginning stating the number of vertices it contains.

Engine

The engine application will receive a configuration file, written in XML. In this phase the XML file will contain the camera settings and the indication of which previously generated files to load.

The screenshots below show a result for each model, created with the examples provided in the generator section.



Example of a XML configuration file for phase one:

```
<world>
    <camera>
        <position x=10 y=10 z=10 />
        <lookAt x=0 y=0 z=0 />
        <up x=0 y=1, z=0 />
        <projection fov=60 near=1 far=1000 />
    </camera>
    <group>
        <models>
            <model file="cone.3d" />
            <model file="plane.3d" />
        </models>
    </group>

</world>
```

Note: in the above example it is assumed that the files “cone.3d” and “plane.3d” have been previously created with the generator application.

The XML file should be read only once when the engine starts. Students should define appropriate data structures to store the information of the model files in memory.

Several alternatives are available to assist the XML Reading, such as tinyXML (http://www.grinninglizard.com/tinyxml/). For other alternatives check out this discussion at stackoverflow (http://stackoverflow.com/questions/170686/what-is-the-best-open-xml-parser-for-c).

Phase 2 – Geometric Transforms

This phase is about creating hierarchical scenes using geometric transforms. Only the engine application needs to be updated. A scene is defined as a tree where each node contains a set of geometric transforms (translate, rotate and scale) and optionally a set of models. Each node can also have children nodes.

Example of a configuration XML file with a single group:

```
<world>
    <camera>
         <position x=10 y=10 z=10 />
         <lookAt x=0 y=0 z=0 />
         <up x=0 y=1, z=0 />
        <projection fov=60 near=1 far=1000 />
    </camera>
    <group>
          <transform>
               <translate x=4  y=0 z =0 />
               <rotate angle=30 x=0 y=1 z=0 />
               <scale x=2 y=0.3 z=1 />
          </transform>
          <models>
               <model file="cone.3d" />
          </models>
     </group>
</world>
```

Example of a hierarchical group definition:

```
<world>
    <camera>
         <position x=10 y=10 z=10 />
         <lookAt x=0 y=0 z=0 />
         <up x=0 y=1, z=0 />
        <projection fov=60 near=1 far=1000 />
     </camera>
     <group>
          <transform>
               <translate x=4  y=0 z =0 />
               <rotate angle=30 x=0 y=1 z=0 />
               <scale x=2 y=0.3 z=1 />
          </transform>
          <models>
               <model file="cone.3d" />
          </models>
          <group>
               <transform>
                    <translate x = 4  y = 0 z  = 0/>
               </transform>
               <models>
                    <model file="sphere.3d" />
```

```
                </models>
                <group>
                     <transform>
                          <translate x = 0  y = 4 z  = 0/>
                     </transform>

                     <models>
                          <model file="cone.3d" />
                     </models>
                </group>
           <group>
     </group>
</world>
```

In the second example the children will inherit the parents geometric transforms.

Geometric transformations can only exist inside a group, and are applied to all models and subgroups.

Note: the order of the geometric transforms is relevant.

The required demo scene for this phase is a static model of the solar system, including the sun, planets and moons defined in a hierarchy.

Due dates

- See blackboard

The deadline is always at midnight. Students may submit work late, except in the last phase, with a penalty of 10% per day. Weekends count as one day.

In all phases, students are required to deliver:

- The source code, Visual Studio project, or makefile, or CMake file.
- All required libraries and include files from third party APIs
- Demo scenes
- A written report detailing the decisions and approaches taken during the phase development.

Assessment: Each phase: 22.5% (17.5% application, 5% written report)

Extras: 10% (examples: third person camera motion, complex demo scenes created for the engine, meaningful XML format extensions, view frustum culling, etc...)