CSE 461 Programming Assignment 1

DUE

April 8, 2025, 23:55

Description

- This is an individual assignment. Please do not collaborate
- If you think that this document does not clearly describes the assignment, ask questions before its too late.

This assignment is about implementing a simple ray tracer.

Scene Description File

This file is in “xml” format. You can use xml parsers.

Definition of the elements which can be found in a scene description file:

- `maxraytracedepth`: This is positive integer which defines the maximum recursive ray tracing depth. Rays starting from camera have depth of 0.
- `background`: `r`, `g`, `b` values for the pixel in a no-hit case. You are going to set the color of the pixel to this value in case the ray does not intersect with any objects in the scene.

```
<background>r g b</background>
```

- `camera`: This defines a camera in the scene. It has the following subfields:
  - `position`: `x,y,z` coordinates of the camera.
  - `gaze`: gaze direction of the camera.
  - `up`: up vector of the camera.
  - `nearplane`: `left`, `right`, `bottom`, `top` values of the image plane.
  - `neardistance`: distance between the image plane and the camera. `gaze` vector is perpendicular to the image plane.
  - `imageresolution`: `nx` and `ny` dimensions of the image.

```
<camera>
    <position>x y z</position>
    <gaze>x y z</gaze>
    <up>x y z</up>
    <nearplane>left right bottom top</nearplane>
    <neardistance> distance </neardistance>
    <imageresolution>nx ny</imageresolution>
</camera>
```

- `ambientlight`: `r`, `g`, `b` ambient light values. This is the amount of light received by the surface which in shadow.

```
<ambientlight>r g b</ambientlight>
```

- `pointlight`: point light source defined by a position vector and an intensity vector.

```
<pointlight id="someid">
    <position> x y z </position>
    <intesity> x y z </intensity>
</pointlight>
```

- `traingularlight`: planar directional light source defined by a triangle. The direction of the light follows the cross product `(vertex1-vertex2) x (vertex1-vertex3)`

```
 <triangularlight id="someotherid">
           <vertex1> x y z </vertex1>
           <vertex2> x y z </vertex2>
```

```
            <vertex3> x y z </vertex3>
            <intensity> x y z </intensity>
</traingularlight>
```

- `material`: This has the following subfields (the values are between `0.0` and `1.0`):
  - `ambient` :ambient light coeffs
  - `diffuse` :diffues light coeffs
  - `specular` :specular light coeffs
  - `mirrorreflactance` :mirrorreflactance coeff
  - `phongexponent` :phong exponent
  - `texturefactor` :texture blend factor. This value is between 0-1. This determines how much the final color effected by the texture color. If texturefactor is 1, the color coming from texture is directly used. Shading will not effect the color. If texturefactor is 0, the texture will not effect the color calculated.

```
<material id="someid">
    <ambient>x y z</ambinet>
    <diffuse>x y z</diffuse>
    <specular>x y z</specular>
    <phongexponent>e</phongexponent>
    <mirrorreflactance>x y z</mirrorreflanctance>
    <texturefactor>t</texturefactor>
</material>
```

- `vertexdata`: lists the `x`, `y`, `z` coordinates of the all the vertices defined in the scene. The vertices are referred by `faces` field under `mesh`

```
<vertexdata>
x1 y1 z1
x2 y2 z2
x3 y3 z3
...
</vertexdata>
```

- `texturedata`: lists `u`, `v` coordinates.

```
<texturedata>
        v1 u1
        v2 u2
        v3 u3
        ...
</texturedata>
```

- `textureimage`: provides the relative name of the texture image file. There is only one texture image file which will be addressed by all the texture coordinates.

```
<textureimage>image_file_name</textureimage>
```

- `normaldata`: defines normal vectors for faces. The format is the same with vertex data section. Multiple faces may refer to the same normal vector.

- `mesh`: Each mesh is a list of faces. Each face is a triangle. Triangles are defined by three vertex indices given in counter-clockwise direction. The vertex index is followed by the uv index after the char `/`. The uv index is followed by the normal index. You can learn more about this organization by reading references about `.obj` file format.

```
<mesh id ="someid">
    <materialid>id</materialid>
    <faces>
        face1_v1_id/face1_t1_id/face1_n1_id face1_v2_id/face1_t2_id/face1_n1_id
↪   face1_v3_id/face1_t3_id/face1_n1_id
        face2_v1_id/face2_t1_id/face2_n1_id face2_v2_id/face2_t2_id/face2_n1_id
↪   face2_v3_id/face2_t3_id/face2_n1_id
```

```
        face3_v1_id/face3_t1_id/face3_n1_id face3_v2_id/face3_t2_id/face3_n1_id
↪   face3_v3_id/face3_t3_id/face3_n1_id
        ...
    </faces>
</mesh>
```

Here is the complete skeleton of the scene description file:

```
<scene>
    <maxraytracedepth>n</maxraytracedepth>
    <background>r g b</background>
    <camera>
        <position>x y z</position>
        <gaze>x y z</gaze>
        <up>x y z</up>
        <nearplane>left right bottom top</nearplane>
        <neardistance> distance </neardistance>
        <imageresolution>nx ny</imageresolution>
    </camera>
    <lights>
        <ambientlight>r g b</ambientlight>
        <pointlight id="someid">
            <position> x y z </position>
            <intesity> x y z </intensity>
        </pointlight>
        <triangularlight id="someotherid">
            <vertex1> x y z </vertex1>
            <vertex2> x y z </vertex2>
            <vertex3> x y z </vertex3>
            <intensity> x y z </intensity>
        </traingularlight>
        ...
    </lights>
    <materials>
        <material id="someid">
            <ambient>x y z</ambinet>
            <diffuse>x y z</diffuse>
            <specular>x y z</specular>
            <phongexponent>e</phongexponent>
            <mirrorreflactance>x y z</mirrorreflanctance>
            <texturefactor>t</texturefactor>
        </material>
        ...
    </materials>


    <vertexdata>
        x1 y1 z1
        x2 y2 z2
        x3 y3 z3
        ...
    </vertexdata>

    <texturedata>
        v1 u1
        v2 u2
        v3 u3
        ...
    </texturedata>

    <textureimage>image_file_name</textureimage>

    <normaldata>
        x1 y1 z1
        x2 y2 z2
        x3 y3 z3
        ...
    </normaldata>


    <objects>
        <mesh id="someid">
            <materialid>id</materialid>
                <faces>
                    face1_v1_id/face1_t1_id/face1_n1_id face1_v2_id/face1_t2_id/face1_n1_id
↪   face1_v3_id/face1_t3_id/face1_n1_id
                    face2_v1_id/face2_t1_id/face2_n1_id face2_v2_id/face2_t2_id/face2_n1_id
↪   face2_v3_id/face2_t3_id/            face2_n1_id
                    face3_v1_id/face3_t1_id/face3_n1_id face3_v2_id/face3_t2_id/face3_n1_id
↪   face3_v3_id/face3_t3_id/            face3_n1_id
                    ...
                </faces>
        </mesh>
        ...
    </objects>
</scene>
```

Below is a simple example scene:

```
<scene>
    <maxraytracedepth>6</maxraytracedepth>
    <backgroundColor>0 0 0</backgroundColor>

    <camera>
        <position>0 0 0</position>
        <gaze>0 0 -1</gaze>
        <up>0 1 0</up>
        <nearPlane>-1 1 -1 1</nearPlane>
        <neardistance>1</neardistance>
        <imageresolution>800 800</imageresolution>
    </camera>

    <lights>
        <ambientlight>25 25 25</ambientlight>
        <pointlight id="1">
            <position>0 0 0 </position>
            <intensity>1000 1000 1000</intensity>
        </pointlight>
        <triangularlight id="2">
            <vertex1> 0 0 0 </vertex1>
            <vertex2> 1.2 0.5 0.5 </vertex2>
            <vertex3> 0.5 0.5 0.5 </vertex3>
            <intensity> 800 800 800 </intensity>
        </traingularlight>
```

```
</lights>

<materials>
    <material id="1">
        <ambient>1 1 1</ambient>
        <diffuse>1 1 1</diffuse>
        <specular>1 1 1</specular>
        <mirrorreflactance>0 0 0</mirrorreflactance>
        <phongexponent>1</phongexponent>
        <texturefactor>0.5</texturefactor>
    </material>
</materials>

<vertexdata>
    1.000000 -1.000000 -1.000000
    1.000000 -1.000000 1.000000
    -1.000000 -1.000000 1.000000
    -1.000000 -1.000000 -1.000000
    1.000000 1.000000 -0.999999
    0.999999 1.000000 1.000001
    -1.000000 1.000000 1.000000
    -1.000000 1.000000 -1.000000
</vertexdata>

  <texturedata>
    1.000000 0.333333
    1.000000 0.666667
    0.666667 0.666667
    0.666667 0.333333
    0.666667 0.000000
    0.000000 0.333333
    0.000000 0.000000
    0.333333 0.000000
    0.333333 1.000000
    0.000000 1.000000
    0.000000 0.666667
    0.333333 0.333333
    0.333333 0.666667
    1.000000 0.000000
</texturedata>

<normaldata>
    0.000000 -1.000000 0.000000
    0.000000 1.000000 0.000000
    1.000000 0.000000 0.000000
    -0.000000 0.000000 1.000000
    -1.000000 -0.000000 -0.000000
    0.000000 0.000000 -1.000000
</normaldata>

<textureimage>my_texture_only_texture.png</textureimage>

<objects>
    <mesh id="1">
        <materialid>1</materialid>
        <faces>
```

```
                2/1/1 3/2/1 4/3/1
                8/1/2 7/4/2 6/5/2
                5/6/3 6/7/3 2/8/3
                6/8/4 7/5/4 3/4/4
                3/9/5 7/10/5 8/11/5
                1/12/6 4/13/6 8/11/6
                1/4/1 2/1/1 4/3/1
                5/14/2 8/1/2 6/5/2
                1/12/3 5/6/3 2/8/3
                2/12/4 6/8/4 3/4/4
                4/13/5 3/9/5 8/11/5
                5/6/6 1/12/6 8/11/6
                2/1/1 3/2/1 4/3/1
                8/1/2 7/4/2 6/5/2
                5/6/3 6/7/3 2/8/3
                6/8/4 7/5/4 3/4/4
                3/9/5 7/10/5 8/11/5
                1/12/6 4/13/6 8/11/6
                1/4/1 2/1/1 4/3/1
                5/14/2 8/1/2 6/5/2
                1/12/3 5/6/3 2/8/3
                2/12/4 6/8/4 3/4/4
                4/13/5 3/9/5 8/11/5
                5/6/6 1/12/6 8/11/6
            </faces>
        </mesh>
        ...
    </objects>
</scene>
```

Remarks

- You have to use C/C++.
- You can use libraries for xml parsing and image read/write.
- Don't submit anything you cannot explain.
- The only object type is mesh type. Don't include code for other object types like sphere, cylinder etc..
- You just have to implement ray-triangle intersection.
- There may be more than 1 material
- There may be more than 1 point light source
- There may be more than 1 mesh.
- It is a good idea to come with an object oriented solution.
- The performance of your solution is important. So, try to write efficient code. You should use threads.
- Discuss about the running time of your implementation. Test using different scenes and measure the render time. The efficiency of your implementation may affect your grade. (you may lose up to 30% if your implementation is not efficient.)
- In order to calculate the shadow properly and avoid numeric errors, you may need to create an offset on surface points where the rays hit. For this, define a constant in your program.

Turn in

- You are going to submit your implementation in a zip file. You will include a documentation about hot to compile and/or run your program.
- You are going to demonstrate the run of your program. It is going to be either through a teams meeting or in a face-to-face meeting.