Computer Graphics — Task 5

3D Rendering

mgr inż. Paweł Aszklar
P.Aszklar@mini.pw.edu.pl

Warsaw, May 25, 2020

1 Assignment

Create a program with a graphical user interface for displaying in the program window a render of a 3D scnene.

The application should allow the user to control a virtual camera which will determine the point of view the scene is observed from. The minimum requirement is to have a camera pointed at a centre of the scene that allow the user to change the distance from the centre and rotate it around X and Y axes.

Use of external libraries is only permitted for the purpose of drawing points and lines on 2D raster and multiplication of matrices and vectors (*Warning!* Initialisation of various transformation matrices should be implemented by you!)

Additional implementation guidelines can be found on my website for software rendering via triangular mesh rasterization (here) with additional notes on texturing (here) or via ray-casting (here).

During the laboratories you will be assigned one of the scene variants described below.

1. **Textured cylinder.** Place in the centre of the scene a triangular mesh of a cylinder. Beside its position each vertex should be provided with so called texture coordinates. A texture will be an external image loaded by the program from a file, that will be overlaid on the object's surface. Texture coordinates describe a position on that image. While drawing each triangle of the mesh projected on the screen, its interior should be filled with a fragment of the texture image delimited by texture coordinates of its vertices.

   Special care needs to be taken when interpolating texture coordinates along the interior of a projected triangle. Also, the cylinder mesh needs to be generated by the program. Its parameters, such as base radius, height or number of subdivisions, can be hardcoded but you should

be able to easily change them. Additionally, to solve the visibility problem, the back face culling algorithm should be employed.

Take a look at triangle mesh rasterization and texturing guidelines. In the first document, however, you can ignore discussion about normal vectors, lighting, Phong shading model and the z-buffer algorithm.

2. **Textured sphere.** Place in the centre of the scene a triangular mesh of a sphere. Beside its position each vertex should be provided with so called texture coordinates. A texture will be an external image loaded by the program from a file, that will be overlaid on the object's surface. Texture coordinates describe a position on that image. While drawing each triangle of the mesh projected on the screen, its interior should be filled with a fragment of the texture image delimited by texture coordinates of its vertices.

   Special care needs to be taken when interpolating texture coordinates along the interior of a projected triangle. Also, the sphere mesh needs to be generated by the program. Its parameters, such as radius or number of subdivisions, can be hardcoded but you should be able to easily change them. Additionally, to solve the visibility problem, the back face culling algorithm should be employed.

   Take a look at triangle mesh rasterization and texturing guidelines. In the first document, however, you can ignore discussion about normal vectors, lighting, Phong shading model and the z-buffer algorithm.

3. **Cylinder shading.** Place in the centre of the scene a triangular mesh of a cylinder. Beside its position each vertex should be provided with a normal vector. In addition to that specify material coefficients of the Phong illumination model for the object and a position of a single white point light source. Those parameters can be hardcoded but simple to change. When drawing each triangle projected on the screen, for each pixel determine it's corresponding position on object's surface, it's normal vector and finally its colour calculated using Phong illumination model.

   Special care needs to be taken when interpolating 3D positions and normal vectors along the interior of a projected triangle. Also, the cylinder mesh needs to be generated by the program. Its parameters, such as radius or number of subdivisions, can be hardcoded but you should be able to easily change them. Additionally, to solve the visibility problem, the back face culling algorithm should be employed.

   Take a look at triangle mesh rasterization guidelines. You can, however, ignore discussion about the z-buffer algorithm.

4. **Sphere shading.** Place in the centre of the scene a triangular mesh of a sphere. Beside its position each vertex should be provided with

a normal vector. In addition to that specify material coefficients of the Phong illumination model for the object and a position of a single white point light source. Those parameters can be hardcoded but simple to change. When drawing each triangle projected on the screen, for each pixel determine it's corresponding position on object's surface, it's normal vector and finally its colour calculated using Phong illumination model.

Special care needs to be taken when interpolating 3D positions and normal vectors along the interior of a projected triangle. Also, the sphere mesh needs to be generated by the program. Its parameters, such as radius or number of subdivisions, can be hardcoded but you should be able to easily change them. Additionally, to solve the visibility problem, the back face culling algorithm should be employed.

Take a look at triangle mesh rasterization guidelines. You can, however, ignore discussion about the z-buffer algorithm.

5. **Scene loading.** Your program should load the objects to display in the scene from a file. The file format should be of your design. Within it for each object you should store information about:

   - object type, which should be either: cylinder, sphere, cuboid or a cone;
   - object size, e.g. base radius and height for cone and cylinder, radius for a sphere, edge lengths for a cuboid;
   - desired mesh density, i.e. a number of subdivisions when approximating the shape with a triangular mesh (not necessary for a cuboid);
   - position and orientation of the object in the scene expressed as a single affine transformation matrix

   Program should generate meshes for the objects and place them in the scene according to the loaded information. When displaying the scene you can draw object wireframes, i.e. it is enough to draw edges of triangles of each mesh projected onto the screen.

   Take a look at triangle mesh rasterization guidelines. The relevant parts involve transformations and modelling, so you can ignore discussion about lighting, normal vectors, z-buffer, back-face culling, triangle filling and Phong illumination model.

6. **Anaglyph stereoscopy.** The scene in your program should consist of several objects of different types (cylinders, cones, shperes and cuboids). Scene itself can be hardcoded, but you should be able to easily change number and types of objects and for each object:

- its size, e.g. base radius and height for cone and cylinder, radius for a sphere, edge lengths for a cuboid;
- desired mesh density, i.e. a number of subdivisions when approximating the shape with a triangular mesh (not necessary for a cuboid);
- position and orientation of the object in the scene expressed as a single affine transformation matrix

Program should generate meshes for the objects and place them in the scene according that information. Draw the scene on a black background. Each object should be drawn as a wireframe, i.e. by only drawing the edges of triangles from its mesh projected on the screen. Each object, however, needs to be drawn twice using two different stereoscopic projection matrices — one for the left and one for the right eye. Assuming the lenses in anaglyph glasses are red on the left and cyan on the right, when drawing a scene from the left eye's perspective you should only modify blue and green channels of the output image. Conversely, when displaying the scene from the right eye's point of view, only the red channel of a pixel should be modified, leaving the other two untouched.

Take a look at triangle mesh rasterization guidelines. The relevant parts involve transformations and modelling, so you can ignore discussion about lighting, normal vectors, z-buffer, back-face culling, triangle filling and Phong illumination model.

7. **Z-Buffer algorithm.** Place near the centre of the scene two cubes, slightly apart from each other. Their sizes, positions and orientations (the latter two described as an affine transformation matrix) can be hardcoded, but you should be able to easily change them. Additionally each of the faces of both cubes should have unique colour. When drawing mesh triangles projected on the screen, you should fill them with the colour of their respective faces. To solve the visibility problem, you should implement the Z-Buffer algorithm.

   Special care needs to be taken when interpolating 3D positions on the surface of objects corresponding to the pixels on the screen inside each projected triangle.

   Take a look at triangle mesh rasterization guidelines.You can, however, ignore discussion about normal vectors, lighting, Phong shading model and back-face culling.

8. **Spheres ray-casting.** Your scene should contain several spheres described by their positions and radii. Additionally place a single white point light and define for each sphere its colour and material coefficients according to Phong illumination model. The scene should be

drawn using ray-casting, i.e. by casting a ray through each pixel of the output image from the camera position. The colour of each pixel should be determined by the closest intersection of its ray with any sphere. Based on the position of the intersection, vector normal to the sphere surface at that point, colour and material coefficients of said sphere, calculate the output colour using Phong illumination model.

Take a look at ray-casting rendering guidelines for more information.