Assignment 2: Rasterization

Handout date: 09/24/2020
Submission deadline: 10/27/2020, 23:59 EST
Demo date: 10/28/2020, Office hour time

This homework accounts for 25% of your final grade.

Goal of this exercise

In this exercise you will implement the following two components:

1. a 2D editor for vector graphics. The editor will allow to draw simple shapes interactively.
2. a 3D editor that allows to prepare 3D scenes composed of multiple 3D objects.

Eigen

In all exercises you will need to do operations with vectors and matrices. To simplify the code, you will use `Eigen`. Have a look at the "Getting Started" page of `Eigen` as well as the Quick Reference page to acquaintain yourselves with the basic matrix operations supported.

OpenGL

In all exercises you will use OpenGL 3.3 with GLSL version 150.

Submission

1. Follow the link (to be sent by email) to create your repository with starter code.
2. Modify the code following the assignment instructions
3. Add a readme in pdf or markdown format as a report of what you did containing a screenshot for each task
4. Push the code into the repository before deadline

Important

For each task below, add at least one image in the readme demonstrating the results. The code that you used for all tasks should be provided.

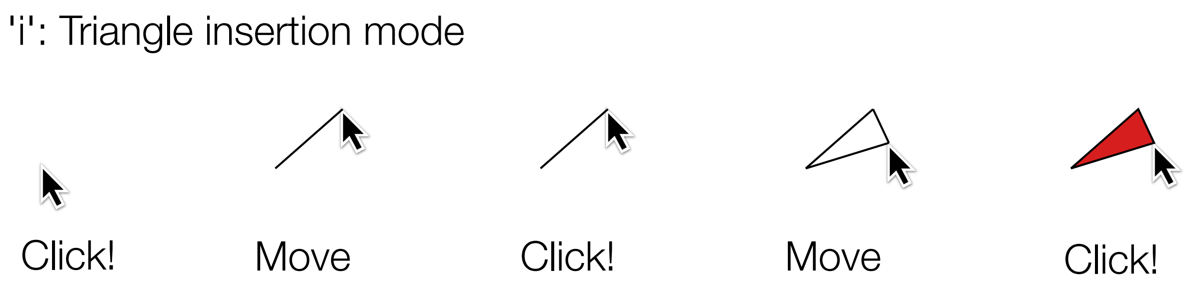
1 2D Editor

These tasks are mandatory. Each one of these tasks is worth 2.0% of the final grade.

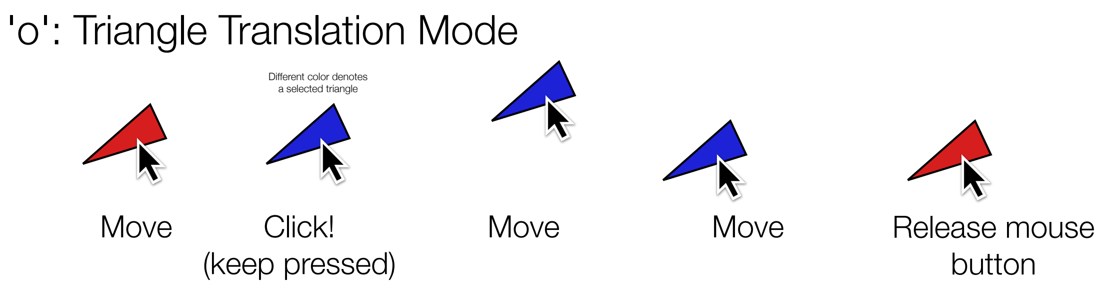
1.1 Triangle Soup Editor

Implement an interactive application that allows to add, edit, and delete triangles. The following operations should be supported:

- The key 'i' will enable triangle insertion mode. When this mode is enabled, every triple of subsequent mouse clicks will create a triangle connecting the three locations where the mouse has been pressed. The first click will create the starting point of the segment, which will be immediately visualized. As the mouse is moved, a preview of a segment will appear. After the second mouse click, a preview of the triangle will appear. After the third click, the current preview will transform into the final triangle.



- The key 'o' will enable triangle translation mode. Each mouse click will selected the triangle below the cursor (which will be highlighted), and every movement of the mouse (while keeping the button pressed) will result in a corresponding translation of the triangle. Note that the triangle should move on screen by the same amount as the cursor.


'o': Triangle Translation Mode
Different color denotes a selected triangle
Move
Click! (keep pressed)
Move
Move
Release mouse button

- The key 'p' will enable delete mode. When the mouse is pressed, the triangle below the cursor is deleted.

1.2 Rotation/Scale

When triangle translation mode is enabled, keep the current primitive selected after the mouse is released. If a primitive is selected and you press the keys 'h' and 'j', the triangle will rotate by 10 degree clockwise or counter-clockwise, respectively. The rotations should be done around its barycenter, i.e. the barycenter of the triangle should not change. When the keys 'k' or 'l' are pressed, the primitive should be scaled up or down by 25%. Similarly to before, the barycenter of the triangle should not move due to the scaling. For this task, you can directly edit the position of the vertices of the triangles on the CPU side, and re-upload them to the GPU after every change.

1.3 Colors

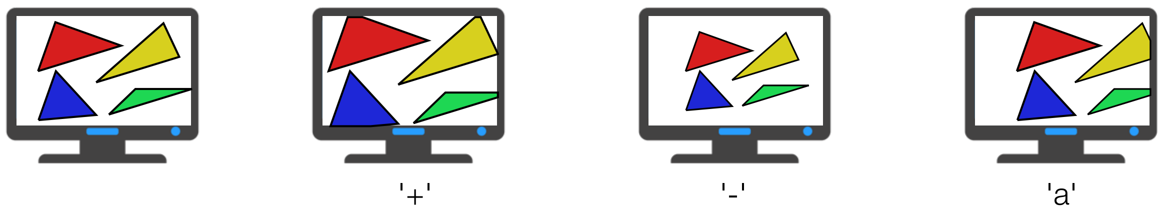
Add the possibility to paint the color of each vertex in the scene. Color mode is enabled by the key 'c'. In this mode, every mouse click will select the vertex closer to the current mouse position. After a vertex is selected, pressing a key from '1' to '9' will change its color (the colors that you use are not important, you can pick whatever colors you like). The color should be interpolated linearly inside the triangles.

1.4 View Control

Add to the application the capability of changing the camera. The following actions should be supported:

- '+' should increase the zoom by 20% zooming in in the center of the screen
- '-' should decrease the zoom by 20% zooming out in the center of the screen
- 'w', 'a', 's', and 'd' should pan the view by 20% of the visible part of the scene, i.e. translate the entire scene, respectively down, right, up and left by 20% of the window size.

This should NOT be implemented by changing the coordinates of the objects in the scene. You must add a view matrix to the vertex shader (as a uniform) that is transforming the position of the vertices of the triangles before they are rendered. Note that you will also have to transform the screen coordinates using the inverse of the view matrix, to ensure that the user interaction will adapt to the current view.



1.5 Add keyframing

Add the possibility to keyframe one property of an object (size, position, or rotation) and create an animation using linear interpolation between the keyframes. You can use a timer to make the animation automatic, or you could move to the next frame at the press of a button.

2 3D Editor

These tasks are mandatory. Each one of these tasks is worth 5.0% of the final grade.

2.1 Scene Editor

Implement an interactive application that allows to add, edit, and delete 3D meshes. The scene should always contain at least one light source. New objects can be added to the scene in three ways:

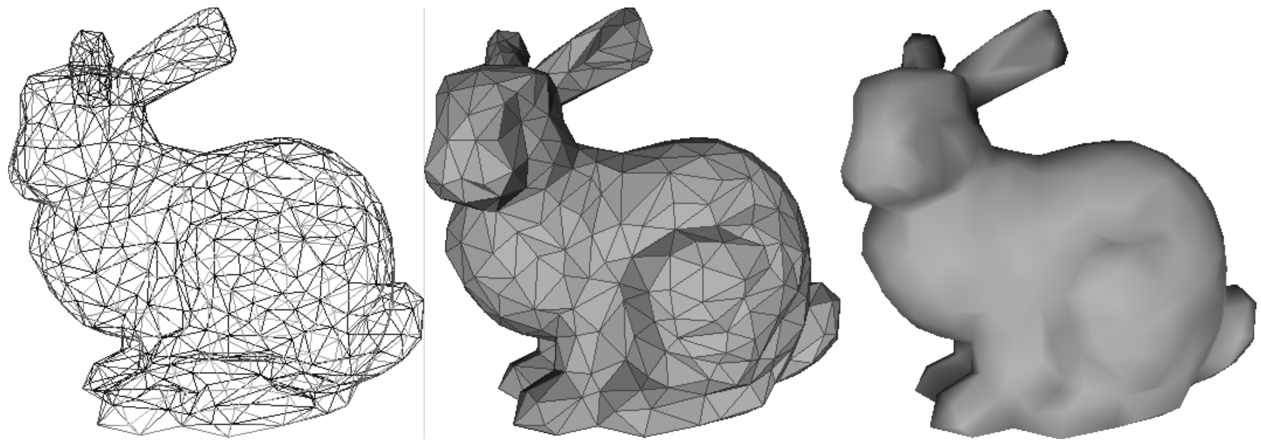
- The key '1' will add a unit cube in the origin
- The key '2' will import a new copy of the mesh *bumpy_cube.off*, scale it to fit into a unit cube and center it on the origin
- The key '3' will import a new copy the mesh 'bunny.off', scale it to fit into a unit cube and center it on the origin

Note that you can have multiple copies of the same object in the scene, and each copy can have its own position, scale, and rotation. For this exercise, all transformations MUST be done in the shader. The VBO containing the vertex positions of each object should be uploaded only once to the GPU.

2.2 Object Control

Clicking on a object will select the object, changing its color. When an object is selected, it should be possible to translate it, rotate it around its barycenter, and rescale it without changing its barycenter. All these actions should be associated to keyboard keys (and the choice of keys should be detailed in the readme). Each object also has a rendering setting associated with it, which can be one of the following three options:

1. Wireframe: only the edges of the triangles are drawn
2. Flat Shading: each triangle is rendered using a unique color (i.e. the normal of all the fragments that compose a triangle is simply the normal of the plane that contains it). On top of the flat shaded triangle, you should draw the wireframe.
3. Phong Shading: the normals are specified on the vertices of the mesh and interpolated in the interior. The lighting equation should be evaluated for each fragment.



To compute the per-vertex normals you should first compute the per-face normals, and then average them on the neighboring vertices. In other words, the normal of the vertex of a mesh should be the average of the normals of the faces touching it. Remember to normalize the normals after averaging.

When an object is selected, it must be possible to switch between the different rendering modes by pressing three keys on the keyboard.

2.3 Camera Control

Add the possibility to translate the position of the camera (similarly to the previous assignment), but in this exercise the camera should always *point to the origin*. It should be possible to move it around, but the camera should always face the origin.

Implement both a *orthographic camera* (similar to the one that you used for Assignment 2, but in 3D) and a *perspective camera*. The cameras should take into account the size of the window, properly adapting the aspect ratio to not distort the image whenever the window is resized. All functionalities should work after resizing the window, including object selection and editing of the scene.

3 Optional Tasks

These tasks are optional. Each one of these taks is worth 3.0% of the final grade.

3.1 Animation Mode

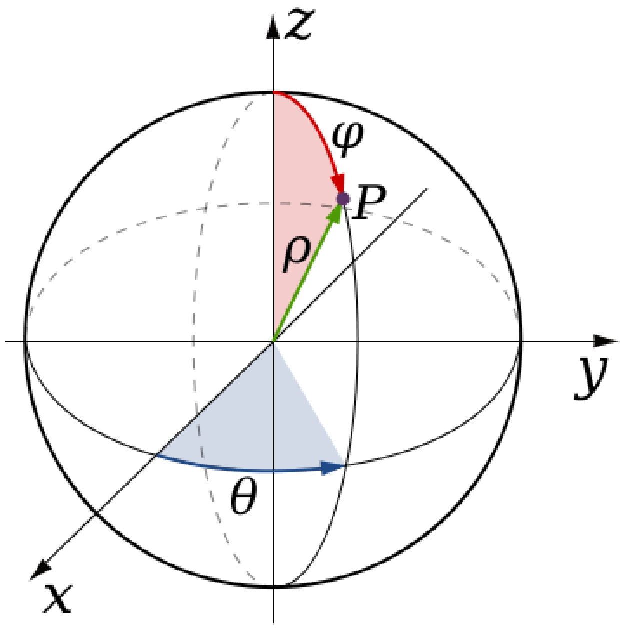
Add the possibility to draw a Bezier curve for each object in the scene. This should be done by having at least one key that allows to add a new control point and three keys to translate the point around. It is not necessary to have a UI that allows to edit the curve. Whenever the button space is pressed, each object should start to move following its associated curve, i.e. it's barycenter should follow the curve.

The animation should last 5 seconds, and at the end all objects should go back to their previous state. The curve should be evaluated using De Casteljau's algorithm.

3.2 Export in SVG format

Add the possibility to export the scene currently drawn on the screen in SVG format `https://en.wikipedia.org/wiki/Scalable_Vector_Graphics`. The exported SVG should be compatible with `https://inkscape.org/`. The triangles should be rendered only using flat shading and they should be deformed according to the current view (orthographic and perspective should both be supported). Hidden triangles should not be rendered (to check for hidden triangles, you can cast three rays to its vertices, and if any vertex is not visible you will simply skip the triangle).

3.3 Trackball



Use a trackball to control the camera. This can be achieved restricting the movement of the camera on a sphere centered on the origin. The easiest way to do it is to parametrize the sphere using spherical coordinates, and to assign keyboard keys to move the camera on the sphere. The camera should always look at the origin.