Assignment 2: Rasterization

Handout date: 10/9/2018
Submission deadline: 10/28/2018, 23:59 EST
Demo date: 10/31/2018

This homework accounts for 17.5% of your final grade.

Goal of this exercise

In this exercise you will implement a 2D editor for vector graphics. The editor will allow to draw simple shapes interactively.

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 create your repository in `https://classroom.github.com/g/SHNFK4PN` with starter code. :)
2. Checkout the repository to your local folder, such as e:/work/cg-hw2.
3. Make a new folder "build", so you will get "e:/work/cg-hw2/build"
4. Put source code of eigen, glfw and glew according to the specified directories in CMakeLists.txt
5. run cmake to generate a visual studio solution in the folder "build".
6. open the solution, compile, run, and see the output images. If there are some link errors, such as error LNK2001: unresolved external symbol __glewBindBuffer, try to link with *lib*
Debug

libglew32d.lib instead of *lib*
Debug
glew32d.lib.

7. Modify the code following the assignment instructions
8. Add a readme in markdown format as a report of what you did with a screen-shot for each task
9. Commit and push the code into the repository before deadline.

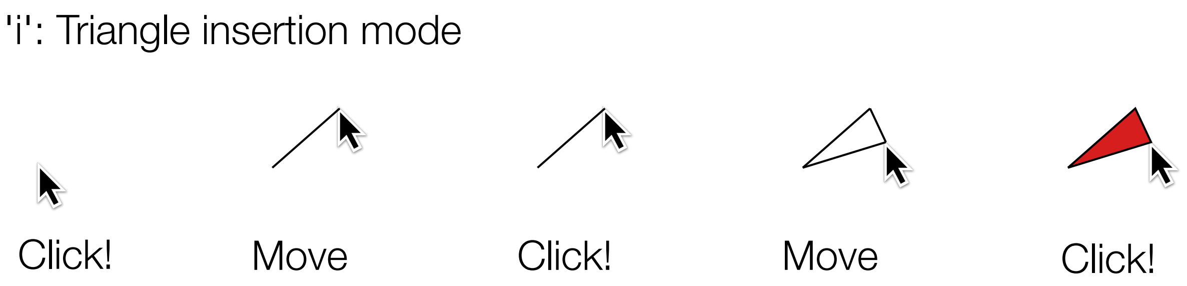
1 Mandatory Tasks

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

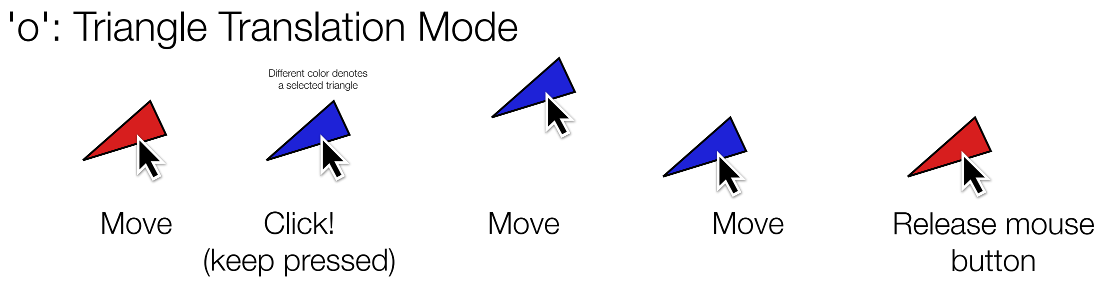
1.1 Triangle Soup Editor

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. If you do it directly in the vertex shader, you can gain additional points (see Task 1.8)

1.3 Colors

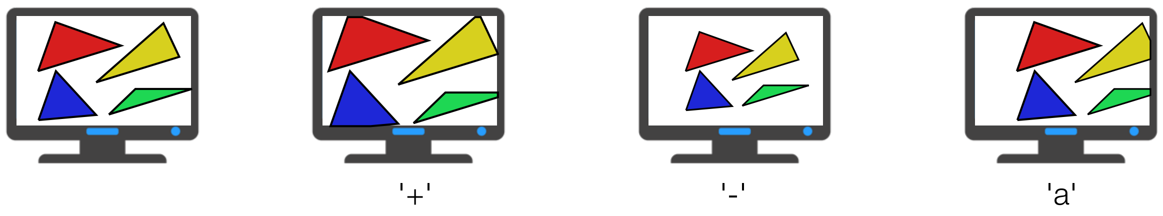
Add the possibility to paint the color of each vertex in the scene. Color mode is enabled by the key 'c'. In this mode, every mouse click will select the vertex closer to the current mouse position. After a vertex is selected, pressing a key from '1' to '9' will change its color (the colors that you 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.

Optional Tasks

These tasks are optional. Each one of these tasks is worth 1.5% of the final grade. The optional points are added to the points of the other exercises, but the total sum of points that you gain with exercises cannot be more than 80%.

1.6 Additional primitive

Add a new mode that allows to add bezier curves and to move their control points.

1.7 Export in SVG format

Add the possibility to export the current drawing in SVG format https://en.wikipedia.org/wiki/Scalable_Vector_Graphics. The exported SVG should be compatible with https://inkscape.org/.

1.8 Shader translation/scaling/rotation

Upload every single triangle as a separate primitive in a separate VBO (or in a single VBO using offsets for drawing them one by one). For each primitive, upload to the GPU its model matrix (as a uniform) that transforms the triangle from its canonical position (defined at its creation) to the current position (obtained by combining translations, scaling and rotations). The transformation should be executed in the shader, and the content of the VBO storing the vertex positions never updated.