CSE 5542, Lab Assignment 1
2D Drawing\(^1\), due on 02/07/2012 11:59PM

1 Goals

In this homework, we will learn how to draw OpenGL primitives in 2D, how to handle GLUT mouse, keyboard and menu events, and how to implement basic 2D transformations manually.

2 Requirements

Please write a 2D OpenGL program that contains the following features.

2.1 2D Viewing (1 Point)

By default, OpenGL/GLSL assumes that the world window spans from \((-1, -1)\) to \((1, 1)\) in the 2D world space, while the viewport spans over the whole screen window. In other words, the function \texttt{gluOrtho2D(left, right, bottom, top)} uses \texttt{left = -1, right = 1, bottom = -1, and top = 1.}

By defining these four values as global variables and modifying them in different event functions, we can achieve simple 2D viewing effects.

**Zoom In and Out (0.5 Point).** Achieve the zoom in effect when the user presses the \texttt{a} key, and the zoom out effect when the user presses the \texttt{z} key. You can do this by scaling the four variables using a scalar \(s\):

\[
\text{left} \leftarrow \text{left} \cdot s; \quad \text{right} \leftarrow \text{right} \cdot s; \quad \text{bottom} \leftarrow \text{bottom} \cdot s; \quad \text{top} \leftarrow \text{top} \cdot s,
\]

in which \(s\) determines whether it’s zoom in \((s < 1)\), or zoom out \((s > 1)\), or no change at all.

**Translation (0.5 Point).** Achieve the translation effect when the user presses \texttt{LEFT, RIGHT, UP,} and \texttt{DOWN} keys. For example, when \texttt{LEFT} is pressed, we update \texttt{left} and \texttt{right} as:

\[
\text{left} \leftarrow \text{left} - t; \quad \text{right} \leftarrow \text{right} - t,
\]

in which \(t\) is the distance you want to translate.

Please define \(s\) and \(t\) by yourself. The sample program has already drawn the two axes from \(-1\) to \(1\). They will help you verify the correctness of your implementation. Remember to call the resizing and drawing functions after you update the four values, so the frame buffer can be redrawn.

2.2 Polyline (5.5 points)

A polyline is a chain of line segments. To define it, we can create a data structure \texttt{POLYLINE}:

\[
\text{POLYLINE} = \\
\{ \\
\quad \text{float points}[512][2]; \\
\quad \text{int point_number}; \\
\};
\]

\(^1\)You may find this document and other sample OpenGL programs at: http://www.cse.ohio-state.edu/~whmin/courses/cse5542-2013-spring/main.html.
Here `points` stores the point data and `point_number` indicates how many points exist in this polyline. To draw this polyline, we can use the OpenGL primitive: `GL_LINE_STRIP`. There can be many polylines in your scene, so you need an array to store them as well. Our goal is to define a set of functions for the user to create such polylines.

**Point Transformation (1.5 Points).** Since the event function receives inputs in the device window space, while the polyline is drawn in the world window, we need to define a function `Transformation` that transforms a point from \((x^D, y^D)\) in the device space to \((x, y)\) in the world space, as Figure 1 shows. The transformation from the device space to the screen space can be done by a simple “flipping” operation, while the transformation from the screen space (or the viewport) to the world space can be formulated as the inverse of the window-to-viewport mapping in the course note.

Whenever the user clicks the left mouse button, use the `Transformation` function to calculate its corresponding position in the world space, and then draw a dot in the world space to verify whether your implementation is correct or not. Use the `glPointSize` function to make the point more visible.

**Polyline Operations (3 points).** Please implement the following functions to create and modify a polyline.

- **Create a polyline.** A new polyline is created when the left mouse button is clicked. The first vertex of this polyline is located where the mouse click happens.

- **Add a vertex.** A new vertex is added to the current polyline, when the user press the f key. The position of this new vertex is calculated as:
  \[
  x = x^{\text{prev}} + u, \quad y = y^{\text{prev}} + v,
  \]
  in which \(u\) and \(v\) are two global offset variables, and \((x^{\text{prev}}, y^{\text{prev}})\) is the previous vertex. You may set the initial values of \(u\) and \(v\) as 0 and 0.04 (or whatever you like).

- **Modifying the last vertex.** By updating \(u\) and \(v\), we can recalculate the last vertex and apply transformations on the last segment of the polyline.
Transformation | Key | Operation
--- | --- | ---
Rotate $10^\circ$ | t | $u_{\text{new}} = u \cos(\pi/18) - v \sin(\pi/18)$, $v_{\text{new}} = u \sin(\pi/18) + v \cos(\pi/18)$
Enlarge 50% | S | $u_{\text{new}} = 1.5u$, $v_{\text{new}} = 1.5v$
Shrink 50% | s | $u_{\text{new}} = 0.5u$, $v_{\text{new}} = 0.5v$

After $u$ and $v$ are updated, recalculate $x$ and $y$. This should get the last segment transformed. Note that we use two temporary variables $u_{\text{new}}$ and $v_{\text{new}}$, rather than updating $u$ and $v$ immediately using the above equations. Why?

- **Assign colors (1 point).** Use an OpenGL popup menu to assign different colors to the current polyline and those polylines created afterwards. Each color is hard-coded as an item in the menu.

### 2.3 Polyline Patterns (3 points)

We can use the basic functionalities in Subsection 2.2 to generate a set of combo polyline patterns. Let $\text{Forward}()$ be the function handling $f$, $\text{Turn}(\theta)$ be the rotation function handling $t$, and $\text{Scale}(s)$ be the scaling function handling $s$ and $S$, we construct the following patterns:

<table>
<thead>
<tr>
<th>Pattern</th>
<th>Key</th>
<th>Procedures</th>
</tr>
</thead>
</table>
| Polygon($n$) | 'p' | for($i=0; i < n; i++$) { $\text{Forward}()$; $\text{Turn}(2\pi/n)$; }
| Star($n$)    | 'a' | for($i=0; i < n; i++$) { $\text{Forward}()$; $\text{Turn}(4\pi/n)$; }
| Spiral($n$, $\theta$, $s$) | 'r' | for($i=0; i < n; i++$) { $\text{Forward}()$; $\text{Turn}(\theta)$; $\text{Scale}(s)$; }

You can choose $n$, $\theta$, and $s$ by yourself. (For $\text{Star}(n)$, $n$ must be a prime number.) After the user presses the key, the corresponding pattern will be added to the current polyline. In practice, the user may want to start a new polyline before adding the pattern, so that it does not mess up with any existing line segment. Please make sure your color menu still works.

### 2.4 Cleanup (0.5 point)

Press the c key to remove all polylines on the screen.

### 3 Bonus Credit (1 point)

When the world window is not centered at the origin (after you did some translations), the zoom in and zoom out operations will cause the center of your world window to change. Can you prevent this from happening?

### 4 Submission Guideline

Please submit the source code (including Visual Studio solution file and project file, but NOT executables or object files) by email to our grader: Xiaoyin Ge (gex AT cse.ohio-state.edu). Use “CSE 5542 Lab 1” as the email subject. The TA office hour is from 3:30 to 5:30PM Friday in DL 474. You can also ask questions on Piazza, especially if you get some immediate questions.