Inf-1100, Assignment 4

Submission: 23:59 October 22nd, 2021

Overview

In this assignment, you will implement C code for rotation and animation.

Description

Your task is first to extend the code from the previous assignment with support for rotating triangles around the Z axis. Then, you should proceed by creating a moving object on the screen (i.e. animation).

Rotation

Rotation around the Z axis can be performed by applying the following formulas to each of the x and y coordinates of a triangle:

$$x' = \cos(r)*x - \sin(r)*y$$

$$y' = \sin(r)*x + \cos(r)*y$$

where r is the degree of rotation. Note that the C library cos and sin functions operate in radians, not degrees. To convert from degree to radians use the following formula:

$$\text{radians} = \text{degree}*\pi/180$$

The C math library defines π as M_PI.

Add support for rotation of triangles by writing a new function in the *triangle.c* file:

void RotateTriangle(triangle_t *triangle)

RotateTriangle should apply the formulas above to rotate the on-screen coordinates of the triangle according to a rotation variable in the triangle data structure (extend the triangle data structure with a float variable called *rotation*). The rotation variable should tell the degree of rotation for the triangle. In **DrawTriangle**, insert a call to **RotateTriangle** just after the **ScaleTriangle** call.

Animation

The illusion of movement is created by displaying a series of images (frames) on the screen where rotation, scale, or translation has been shifted slightly in each image.

One of the problems that you must solve is how to ensure that the rate of animation (i.e. the time it takes for the transition from point A to point B on the screen) is independent of the speed of the underlying CPU. For example, assume your program just consists of a loop that continuously updates animation variables (rotation, scale, etc.), renders, and visualizes animation frames. With this approach, the time it takes to, say rotate an object 90 degrees, is totally dependent upon the speed of the underlying CPU. A fast CPU might perform the

90-degree rotation in 0.1s, while a slow CPU might require 0.5s. We suggest solving this problem using one of the following approaches:

- Decide on a fixed number of animation updates per second, say 50 (i.e. one new update per 1000/50=20 milliseconds). Then measure the time it takes to calculate and display one animation frame (using SDL_GetTicks), and delay the creation of the next frame until 20ms of time has passed (using SDL_Delay). A drawback with this approach is that the animation will slow down if there is not enough processing power to create animation frames at the target rate.
- An alternative approach is to introduce partial animation updates. Assume you decide on 50 animation updates per second and, similar to the above approach, you measure the time it takes to create one animation frame. Based on this time you calculate how many animation updates to perform. For example, if it took 10ms to create the frame, the animation variables should receive 10/20=0.5 updates. Similarly, if it took 30ms to render the frame, the animation variables should receive 30/20=1.5 updates. Using this approach, the animation will progress at a fixed rate. A drawback, however, is that the animation will seem 'jerky' if there is insufficient CPU available.

C solution

When all students have handed in their answers to the previous assignment we will distribute a set of files that you can use as a starting point for this assignment. This set of files, however, will not contain any more functionality than the code you developed for the previous assignment. We suggest basing your work on this assignment on the code you developed for the previous assignment.

Submission

This assignment is not mandatory. However, we highly recommend you complete it.