CSD2341 Computer Graphics Programming

Workshop 5: Geometric Transformations

Assessment: 6 Marks. Due in the next workshop.
For on-campus students: demonstrate in the next workshop session.
For off-campus students only: zip your project directory and submit to blackboard.

Related Objectives from the Unit outline:

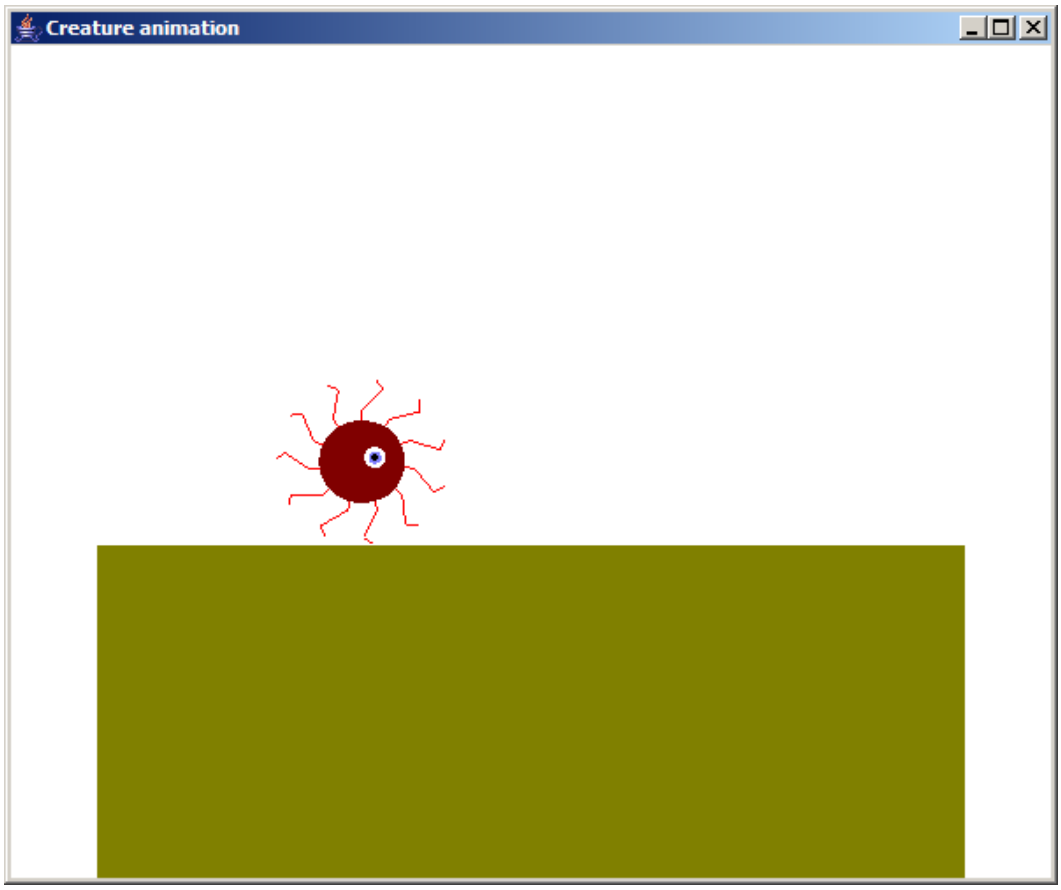
- understand the programming techniques and algorithms used for developing 2D graphics software.

Learning Outcomes:

In this workshop, you will learn how to use geometric transformation in openGL.

Geometric transformations can be used to construct compound objects from simpler ones, and to move objects around a scene in an animation. In this workshop, you will use transformations in openGL to animate a strange creature.

The creature looks like this:



It has a round body, one eye, and 11 legs which it spins around its body to move itself along. In the example above, it is moving from right to left.

Your aim is to have the creature walk along the platform from left to right, turn, and walk from right to left, turn, and so on.

1. Download the file `AnimateCreature.zip` from BlackBoard, and unzip it somewhere. Inside the folder there should be a folder called `src` containing 3 java files.
2. Set up a new NetBeans project using the unzipped folder as the project folder, and `src` as the source folder. It contains three files: `AnimateCreature.java`, `Creature.java` and `Axes.java`.

   `AnimateCreature.java` is the driver program. It opens a JOGL display window, creates a creature, and then repeatedly calls the creature's `update()` method to make the creature move, and its `draw()` method to draw each frame of the animation.

   `Creature.java` is not complete as yet --- you have to complete it. To start with, it has no eye, and only one leg, on which it slides along. You will add more functionality a little at a time.

`Axes.java` is a utility class that draws a set of axes at the origin of the current coordinate system.

3. First, compile and run `AnimateCreature`. Remember that you need to add in the external jar files to the project as in earlier workshops.
4. Looking at the `draw()` method in `Creature.java`, you can see that the lines:

```
gl.glTranslated(creatureX, ground, 0);
gl.glScaled(width/2, height/2, 1);
```

are used to move the creature's coordinate system so that the creature will be drawn in the right location in world coordinates, and then to scale the coordinate system so that the creature will be drawn the right size.

To see the effect of these transformations, you could change the value of the `boolean SHOW_AXES` to `true`.

There are a number of different coordinate systems being used in this program. First, the world coordinate system is defined in `AnimateCreature.java` with the call to its superclass (`SimpleGLFrame`) constructor. This is a coordinate system whose (0, 0) is at to the bottom left of the display area, and (600, 500) at the top right, by the viewing transformation.

The creature is defined in its own coordinate system, so that it occupies a square with bottom left at (-1, -1) and top right at (1, 1). The two transformations above transform this into a rectangle with bottom left at `(creatureX – width/2, ground – height/2)` and top right at `(creatureX + width/2, ground + height/2)`. Another transformation inside the `drawCreature()` method moves this rectangle up a bit so that the creature stands on the ground (or just above it - more about this later). The `update()` method changes the variable `creatureX` (and other things), which makes the creature move left or right.

Notice the calls

```
gl.glPushMatrix();
```

and

```
gl.glPopMatrix();
```

at the start and end of the method. These ensure that the graphics state (at

least in terms of the transformation matrix) is restored to the state it was in before the method was called. This is a tidy way to do things that avoids confusing and inconvenient side effects. In the same way, it is good practice to use `glPushAttributes()` and `glPopAttributes()` whenever you need to change drawing attributes, such as colours.

5. Now let's look inside `drawCreature()`. Since the two transformations in `draw()` take care of position and scale, within `drawCreature`, we can forget about those things and draw in local "creature" coordinates. Notice the call:

```
gl.glTranslated(0, 1, 0);
```

   Without this call, the creature would be drawn with the middle of its body at ground level. The call moves the creature up by 1 unit in its local coordinate system, i.e. so that it will be standing on the ground.

6. The call

```
drawLegs(gl);
```

   draws the legs (only one at the moment) in the creature coordinate system. Have a look at the code for this. It just uses some lines.

7. The call

```
gl.glScaled(BODY_SIZE, BODY_SIZE, 1);
```

   controls the size of the creatures body, and is specified in the creature coordinate system.

   After this, `drawBody()` is called to draw the body, and since position and scale have already been taken care of by earlier transformations, `drawBody()` can draw using its own coordinate system (again the body is drawn inside a square with corners (-1, -1) and (1, 1).

8. Now let's look at `drawBody()`. Notice the call

```
glu.gluDisk(circle, 0, 1, 20, 1);
```

   This is one way to draw a filled circle in openGL. It is drawn with centre at (0, 0). The parameters are:

   `circle`: a `GLUquadric`,
   0: internal radius
   1: external radius
   20: slices
   1: rings

This method can be used to draw a disc shape, with a hole in the middle (that's what the internal radius is for). Slices and rings affect the quality of the circle (try changing them to see).

There are a couple of other techniques illustrated in this program:

a. In `AnimateCreature` you will see the lines

```
gl.glEnable(GL.GL_BLEND);
gl.glBlendFunc(GL.GL_SRC_ALPHA,
GL.GL_ONE_MINUS_SRC_ALPHA);
```

These lines enable blending and defines how colours should be combined when they are blended. This allows the drawing of transparent objects (you'll see them if you set `SHOW_AXES` to true).

b. In `Creature.java`, there is a display list used to draw a leg. Display lists can be used to "pre-compile" a common sequence of drawing operations, making the drawing more efficient.

Exercises

a. Add an eye to the body (1 Mark);
b. Have the creature face the way it is moving (1 Mark);
c. Add more legs (1 Mark);
d. Spin the legs in the direction of movement (1 Mark);
e. Make the creature jump up when it turns around (1 Mark);
f. Make the creature squat to get ready to jump (1 Mark).

Implement and test each one as you go. If you get stuck, there are some hints at the end, and you can ask your tutor.

9. Adding an eye.

a. First write a method `drawEye(GL gl)`, that draws an eye in its own coordinate system, within a square with bottom left at (-1, -1) and top right at (1, 1). Set the appropriate colours using `gl.glColor3d()` and use `glu.gluDisk()` to draw a filled circle(s). Remember to use `gl.glPushAttributes()` and `gl.glPopAttributes()`.

b. Add private static variables to specify the size of the eye and its position within the body coordinate system.

c. Add a call to `drawEye()` in `drawBody()`, after suitable translation and scaling transformations to give the eye the right location and size within the body.

   Try to do this yourself. If you get stuck, check Hint #1 at the end of this document.

10. Have the creature face the way it is moving.

    a. Notice that there is a variable `dir`, which tells whether the creature is facing left or right.

    b. In the `drawCreature()` method, before any other transformations or drawing are done, add a transformation to flip the creature about its own y-axis if it is facing left. Once this is done, we can always draw the creature facing to the right.

       If you get stuck, check Hint #2.

11. Add more legs.

    a. At the moment, only one leg is drawn, by calling `drawLeg()`. It goes straight down to the ground.

    b. We can draw another leg by rotating the coordinate system a little, and then calling `drawLeg()` again.

    c. Add a loop that calls `drawLeg()` and then rotates the coordinate system, each time around the loop. Define a private static variable that specifies how many legs the creature has, and use this variable to determine how many times to go around the loop, and how much to rotate each time.

       If you get stuck, check Hint #3.

12. Spin the legs in the direction of movement.

    a. This is easy. Add a rotation transformation before the call to `drawLegs()`. The variable `legAngle` gives you the amount of rotation to use.

       If you get stuck, check Hint #4.

13. Make the creature jump when it turns around.

    a. The amount to jump is specified by the variable `elevation`.

    b. Modify the call:

```
gl.glTranslated(0, 1, 0);
```

to raise the creature above the ground instead of just on the ground.

If you get stuck, check Hint #5.

14. Make the creature squat to get ready to jump.
    a. Add a call to scale the creature in the y-direction by a factor of `1.0-squat`.

       If you get stuck, check Hint #6.
15. Answer the Tutorial Questions over the page.
16. Demonstrate the assessed parts of the workshop next week, or for off-campus students, Zip up your solution, and submit using BlackBoard, by the next workshop.

Solutions will be posted on BlackBoard for both the programming task and the tutorial questions.

Tutorial questions on Geometric Transformations

1. Construct a transformation matrix to rotate a point 80° anticlockwise, and then scale it by a factor of 2 in the x direction and 4 in the y direction. (You can use the following values: cos(80°) = 0.174, sin(80°) = 0.985.) (2 marks)

2. Use it to transform the triangle with vertices at (2, 5), (4, 7) and (6, 2). (2 marks)

3. Answer this question when you have completed the workshop.
   a. In `draw()`, we apply the following transformations:

      ```
      gl.glTranslated(creatureX, ground, 0);
      gl.glScaled(width/4, height/2, 1);
      ```

      Supposing that `creatureX` is 30, `ground` is 80, and `width` and `height` are both 100, what will the transformation matrix be after these calls? Show your working. (1 mark).

   b. After this, in `drawCreature()`, we apply the following:

      ```
      if(dir == Direction.LEFT)
      {
           gl.glScaled(-1, 2, 0);
      }
      ```

      Supposing that `dir` is `Direction.LEFT`, what will the composite transformation matrix be now? Show your working. (1 mark).

   c. Then the following transformation is applied:

      ```
      gl.glScaled(1, 1-squat, 1);
      ```

      Supposing that `squat` is 0.2, what will the matrix be now? Show your working. (1 mark).

   d. The centre of the creature is at (0, 0) in its own coordinate system. Where is this point mapped to by the matrix above in the world coordinate system? Show your working. (1 mark).

4. List the steps to determine the transformation matrix to reflect an object about the line y=x+1. Write out the matrices. You don't have to multiply them out. (2 marks)

Hints

Hint #1:

For 10 c(), add these lines to `drawBody()`:

```
gl.glPushMatrix();

// the eye is a little off centre
gl.glTranslated(EYE_X, EYE_Y, 0);
// this determines the size of the eye
// relative to the body
gl.glScaled(EYE_SIZE, EYE_SIZE, 1);
drawEye(gl);

gl.glPopMatrix();
```

Hint #2:

For 11, add the following to `drawCreature()`:

```
// this transformation flips the
// creature around its own y-axis
// so we can just work out how to
// draw it facing right
if(dir == Direction.LEFT)
{
     gl.glScaled(-1, 1, 1);
}
```

Hint #3:

For 12 (c), replace the call to `drawLeg()` in `drawLegs()` with:

```
gl.glPushMatrix();

for(int i = 0; i < NLEGS; i++)
{
     drawLeg(gl);
     // rotate the next leg a bit further
     // so the legs will point in all directions
     gl.glRotated(360.0/NLEGS, 0, 0, 1);
}

gl.glPopMatrix();
```

Hint #4:

In 13, replace the call to `drawLegs()` in `drawCreature()` with:

```
gl.glPushMatrix();

// this transformation makes the
// legs spin around when the creature walks
gl.glRotated(legAngle, 0, 0, 1);
drawLegs(gl);

gl.glPopMatrix();
```

Hint #5:

In 14, change the call to

```
// this one is determines how high off
// the ground the creature will be
// when elevation == 0, the creature
// is on the ground
gl.glTranslated(0, 1+elevation, 0);
```

Hint #6:

In 15, add the call below to `drawCreature()`:

```
// this transformation is used when the creature
// squats down to get ready to jump
gl.glScaled(1.0, 1.0-squat, 1);
```