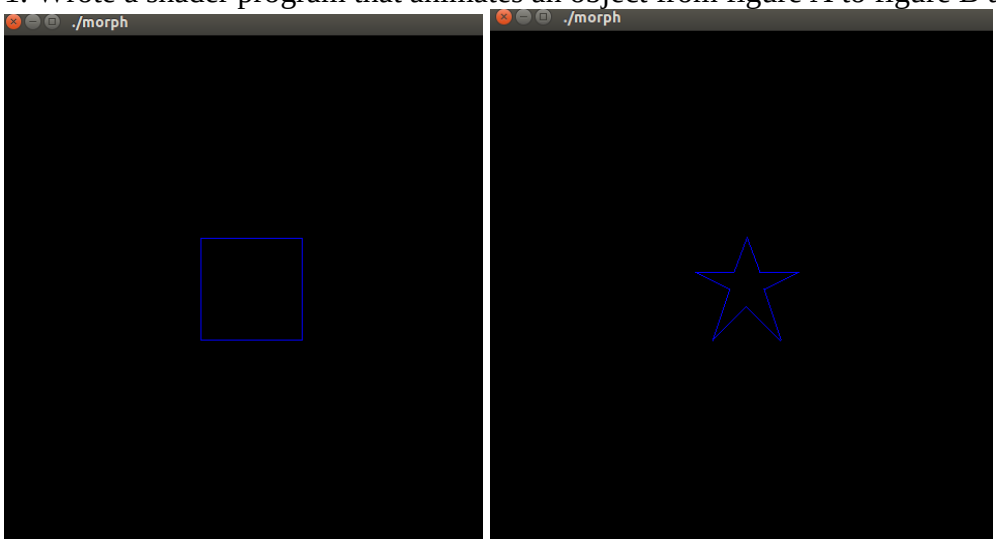
Erik Anchondo
2-1-19
cse 520
homework 1

1. Wrote a shader program that animates an object from figure A to figure B and back.



Here is the source code to the cpp file:

```
GLuint programObject = 0;
GLuint vertexShaderObject = 0;
GLuint fragmentShaderObject = 0;
static GLint win = 0;
GLuint tParam;
float tValue;
GLuint Vertex2Param;
int readShaderSource(char *fileName, GLchar **shader )
{
   FILE *fp;
   int count, pos, shaderSize;
   fp = fopen( fileName, "r");
   if ( !fp ){return 0;}
   pos = (int) ftell ( fp );
   fseek ( fp, 0, SEEK_END );
   shaderSize = ( int ) ftell ( fp ) - pos;
   fseek ( fp, 0, SEEK_SET );
   if ( shaderSize <= 0 )
   {
      printf("Shader %s empty\n", fileName);
      return 0;
   }
   *shader = (GLchar *) malloc( shaderSize + 1);
   count = (int) fread(*shader, 1, shaderSize, fp);
```

```
    (*shader)[count] = '\0';
    if (ferror(fp)){count = 0;}
    fclose(fp);
    return 1;
}
int installShaders(const GLchar *vertex, const GLchar *fragment)
{
    GLint  vertCompiled, fragCompiled;  // status values
    GLint  linked;
    vertexShaderObject = glCreateShader(GL_VERTEX_SHADER);
    fragmentShaderObject = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(vertexShaderObject, 1, &vertex, NULL);
    glShaderSource(fragmentShaderObject, 1, &fragment, NULL);
    glCompileShader(vertexShaderObject);
    glGetShaderiv(vertexShaderObject, GL_COMPILE_STATUS, &vertCompiled);
    glCompileShader( fragmentShaderObject );
    glGetShaderiv( fragmentShaderObject, GL_COMPILE_STATUS, &fragCompiled);
    if (!vertCompiled || !fragCompiled){ return 0; }
    programObject = glCreateProgram();
    glAttachShader( programObject, vertexShaderObject);
    glAttachShader( programObject, fragmentShaderObject);
    glLinkProgram(programObject);
    glGetProgramiv(programObject, GL_LINK_STATUS, &linked);
    if (!linked){ return 0;}
    glUseProgram(programObject);
    GLchar log[1000];
    GLsizei len;
    glGetShaderInfoLog(vertexShaderObject, 1000, &len, log);
    printf("Vert Shader Info Log: %s\n", log);
    glGetProgramInfoLog(programObject, 1000, &len, log);
    printf("Program Info Log: %s\n", log);
    return 1;
}
int init(void)
{
  const char *version;
  GLchar *VertexShaderSource, *FragmentShaderSource;
  int loadstatus = 0;
  version = (const char *) glGetString(GL_VERSION);       //version check
  if (version[0] < '2' || version[1] != '.')
  {
        printf("This program requires OpenGL >= 2.x, found %s\n", version);
        exit(1);
  }
  readShaderSource( (char *) "morph.vert", &VertexShaderSource );         //link vert file
  readShaderSource((char *) "tests.frag", &FragmentShaderSource );        //link frag file
```

```
  loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);
  printf("loadstatus=%d\n", loadstatus );
  tParam = glGetUniformLocation ( programObject, "t" );
  Vertex2Param = glGetAttribLocation ( programObject, "Vertex2" );
  return loadstatus;
}
static void Reshape(int width, int height)
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-5.0, 5.0, -5.0, 5.0, 5.0, 25.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0f, 0.0f, -15.0f);
}
void CleanUp(void)
{
  glDeleteShader(vertexShaderObject);
  glDeleteShader(fragmentShaderObject);
  glDeleteProgram(programObject);
  glutDestroyWindow(win);
}
static void Idle(void) {}
static void Key(unsigned char key, int x, int y)
{
  switch(key)
  {
        case 27:
              CleanUp();
              exit(0);
              break;
        case 'a':
              if ( tValue < 1 ){ tValue += 0.05; }
              glUniform1f( tParam, tValue );
              break;
        case 's':
              if ( tValue >= 0.05 ){ tValue -= 0.05; }
              glUniform1f( tParam, tValue );
              break;
  }
  glutPostRedisplay();
}
void makeFigures( float A[][3], float B[][3] )
{
        //cube
```

```
//          x axis                  y axis          x axis                  y axis
        A[0][0] = 0;            A[0][1] = 3;    A[1][0] = 3;            A[1][1] = 3;
        A[2][0] = 3;            A[2][1] = 1;    A[3][0] = 3;            A[3][1] = -3;
        A[4][0] = 2;            A[4][1] = -3;   A[5][0] = 0;            A[5][1] = -3;
        A[6][0] = -2;           A[6][1] = -3;   A[7][0] = -3;           A[7][1] = -3;
        A[8][0] = -3;           A[8][1] = 1;    A[9][0] = -3;           A[9][1] = 3;
        //star
        B[0][0] = 0;            B[0][1] = 3;    B[1][0] = 0.75;         B[1][1] = 1;
        B[2][0] = 3;            B[2][1] = 1;    B[3][0] = 1;            B[3][1] = 0;
        B[4][0] = 2;            B[4][1] = -3;   B[5][0] = 0;            B[5][1] = -1;
        B[6][0] = -2;           B[6][1] = -3;   B[7][0] = -1;           B[7][1] = 0;
        B[8][0] = -3;           B[8][1] = 1;    B[9][0] = -0.75;        B[9][1] = 1;
}
void display(void)
{
  GLfloat vec[4];
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glClearColor( 0.0, 0.0, 0.0, 0.0 );    //get black background color
  glColor3f ( 1, 0, 0 );                 //color for object, has no effect with shaders
  glPolygonMode( GL_FRONT, GL_LINE );
  glPolygonMode( GL_BACK, GL_LINE );
  int N = 10;
  GLfloat A[N][3], B[N][3];
  makeFigures( A, B );
  glUniform1f ( tParam, tValue );
  glBegin ( GL_POLYGON );
        for ( int i = 0; i < N; ++i )
        {
                glVertexAttrib3fv ( Vertex2Param,  &B[i][0] );
                glVertex2fv ( A[i] );
        }
  glEnd();
  glutSwapBuffers();
  glFlush();
}
int main(int argc, char *argv[])
{
  int success = 0;
  glutInit(&argc, argv);
  glutInitWindowPosition( 0, 0);
  glutInitWindowSize(500, 500);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  win = glutCreateWindow(argv[0]);
  glutReshapeFunc(Reshape);
  glutKeyboardFunc(Key);
  glutDisplayFunc(display);
```

```
  glutIdleFunc(Idle);
   glewInit();
  success = init();
  if ( success ){ glutMainLoop();}
  return 0;
}
```

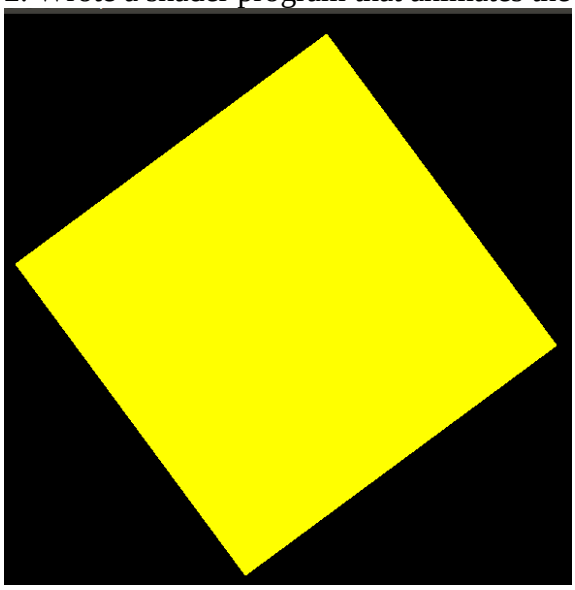
Here is the vert. file:

```
uniform float t;
attribute vec4 Vertex2;
void main(void)
{Vertex2*t
        vec4 f = vec4 ( mix(gl_Vertex.xy, Vertex2.xy, t ), 0.0, 1.0 );
        gl_Position = gl_ModelViewProjectionMatrix * f;
}
```

Here is the frag. file:

```
void main(void)
{
        gl_FragColor = vec4( 0, 0, 1, 0);       //blue color
}
```

2. Wrote a shader program that animates the rotation of a square on its z axis



Here is the source code to the cpp file:

```
GLuint programObject = 0;
GLuint vertexShaderObject = 0;
GLuint fragmentShaderObject = 0;
static GLint win = 0;
int cLoc;
float theta[3];
```

```
int  thetaLoc[3];
int  aLoc;        //holds attribute of a
int  txLoc;       //holds attribute of tx
float tx = 0.0;
int timeCounter = 0;    // time counter
int time_interval = 10;// how much time has to be before action is done
float x = 0, y = 0, z = 0;
bool test = true;
float animation_speed = 0.05;
int readShaderSource(char *fileName, GLchar **shader )
{
   FILE *fp;
   int count, pos, shaderSize;
   fp = fopen( fileName, "r");
   if ( !fp ){ return 0;}
   pos = (int) ftell ( fp );
   fseek ( fp, 0, SEEK_END );
   shaderSize = ( int ) ftell ( fp ) - pos;
   fseek ( fp, 0, SEEK_SET );
   if ( shaderSize <= 0 )
   {
      printf("Shader %s empty\n", fileName);
      return 0;
   }
   *shader = (GLchar *) malloc( shaderSize + 1);
   count = (int) fread(*shader, 1, shaderSize, fp);
   (*shader)[count] = '\0';
   if (ferror(fp)){count = 0;}
   fclose(fp);
   return 1;
}
void checkError()
{
   char *infoLog;
   int  infoLen, len;
   unsigned int shaders[2] = {vertexShaderObject,  fragmentShaderObject};
   char names[][50] = {"Vertex shader", "Fragment shader"};
   for ( int i = 0; i < 2; i++ )
   {
     glGetShaderiv( shaders[i], GL_INFO_LOG_LENGTH, &infoLen);
     if ( infoLen > 0 )
     {
          infoLog = (char *) malloc ( infoLen + 1 );
          glGetShaderInfoLog ( shaders[i], infoLen, &len, infoLog );
          printf("\n***** %s: %s\n", names[i], infoLog );
          delete infoLog;
```

```
    }
   }
}
int installShaders(const GLchar *vertex, const GLchar *fragment)
{
   GLint  vertCompiled, fragCompiled;  // status values
   GLint  linked;
   vertexShaderObject = glCreateShader(GL_VERTEX_SHADER);
   fragmentShaderObject = glCreateShader(GL_FRAGMENT_SHADER);
   glShaderSource(vertexShaderObject, 1, &vertex, NULL);
   glShaderSource(fragmentShaderObject, 1, &fragment, NULL);
   glCompileShader(vertexShaderObject);
   glGetShaderiv(vertexShaderObject, GL_COMPILE_STATUS, &vertCompiled);
   glCompileShader( fragmentShaderObject );
   glGetShaderiv( fragmentShaderObject, GL_COMPILE_STATUS, &fragCompiled);
   checkError();
   printf("vertCompiled, fragCompiled: %d, %d\n", vertCompiled, fragCompiled);
   if (!vertCompiled || !fragCompiled){return 0;}
   programObject = glCreateProgram();
   glAttachShader( programObject, vertexShaderObject);
   glAttachShader( programObject, fragmentShaderObject);
   glLinkProgram(programObject);
   glGetProgramiv(programObject, GL_LINK_STATUS, &linked);
   printf("linked=%d\n");
   if (!linked){ return 0;}
   glUseProgram(programObject);
   return 1;
}
int init(void)
{
  const char *version;
  GLchar *VertexShaderSource, *FragmentShaderSource;
  int loadstatus = 0;
  version = (const char *) glGetString(GL_VERSION);
  if (version[0] < '2' || version[1] != '.')
  {
    printf("This program requires OpenGL > 2.x, found %s\n", version);
    exit(1);
  }
  printf("version=%s\n", version );
  readShaderSource((char *) "template.vert", &VertexShaderSource );
  readShaderSource( (char *) "template.frag", &FragmentShaderSource );
  loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);
  char names[][20] = {"color0", "color1", "color2"};
  int loc[3];
  for ( int i = 0; i < 3; i++ )
```

```
  {
   loc[i] = glGetUniformLocation (programObject, names[i] );
   if ( loc[i] < 0 )
   {
    cout << "No such name: " << names[i] << endl;
   }
  }
  glUniform4f ( loc[0], 1.0, 0, 0, 1 );
  glUniform4f ( loc[1], 0.0, 1.0, 0.0, 1 );
  glUniform4f ( loc[2], 0.0, 0, 1.0, 1 );
  char names1[][20] = { "ax", "ay", "az" };
  for ( int i = 0; i < 3; i++ ) {
   thetaLoc[i] = glGetAttribLocation ( programObject, names1[i] );
   if ( thetaLoc[i] < 0 ) {
    cout << "No such name: " << names[i] << endl;
   }
  }
  aLoc = glGetAttribLocation ( programObject, "a" );          //set a to aloc
  txLoc = glGetAttribLocation ( programObject, "tx" );        //set tx to txLoc
  return loadstatus;
}
static void Reshape(int width, int height)       //FOV
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-1.0, 1.0, -1.0, 1.0, 5.0, 25.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0f, 0.0f, -15.0f);
}
void CleanUp(void)    //for closing program
{
  glDeleteShader(vertexShaderObject);
  glDeleteShader(fragmentShaderObject);
  glDeleteProgram(programObject);
  glutDestroyWindow(win);
}
static void Idle(void)  //this controls timer for animation
{
        if (timeCounter < time_interval)
        {
                timeCounter++;
        }
        else
        {
```

```
                timeCounter = 0;
                x += animation_speed;
                y += animation_speed;
                z += animation_speed;
        }
  glutPostRedisplay();
}
static void Key(unsigned char key, int x, int y)
{
  switch(key)
  {
        case 27:
                CleanUp();      //call clean up function
                exit(0);//close program
                break;
  }
  glutPostRedisplay();
}
void display(void)
{
  GLfloat vec[4];
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glClearColor( 0.0, 0.0, 0.0, 1.0 );    //get black background color
  glColor3f ( 0, 0, 1 );                 //red, this will have no effect if shader is loaded
  cLoc = glGetAttribLocation ( programObject, "c" ); //sets object for .vert shader
  for ( int i = 0; i < 3; i++ )
  {
    glVertexAttrib1f ( thetaLoc[i], theta[i] );
  }
  glVertexAttrib3f ( aLoc, x, y, z );
  glVertexAttrib1f ( txLoc, tx );
  glBegin(GL_QUADS);
        glVertex3f(2.0, 2.0, 0.0);
        glVertex3f(2.0, -2.0, 0.0);
        glVertex3f(-2.0, -2.0, 0.0);
        glVertex3f(-2.0, 2.0, 0.0);
  glEnd();
  glutSwapBuffers();
  glFlush();
}
int main(int argc, char *argv[])
{
  int success = 0;
  glutInit(&argc, argv);
  glutInitWindowPosition( 0, 0);
  glutInitWindowSize(600, 600);
```

```
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  win = glutCreateWindow("lab2 part1");
  glutReshapeFunc(Reshape);
  glutKeyboardFunc(Key);
  glutDisplayFunc(display);
  glutIdleFunc(Idle);
  glewInit();

  success = init();
  printf("success=%d\n", success );
  if ( success ){glutMainLoop();}
  return 0;
}
```

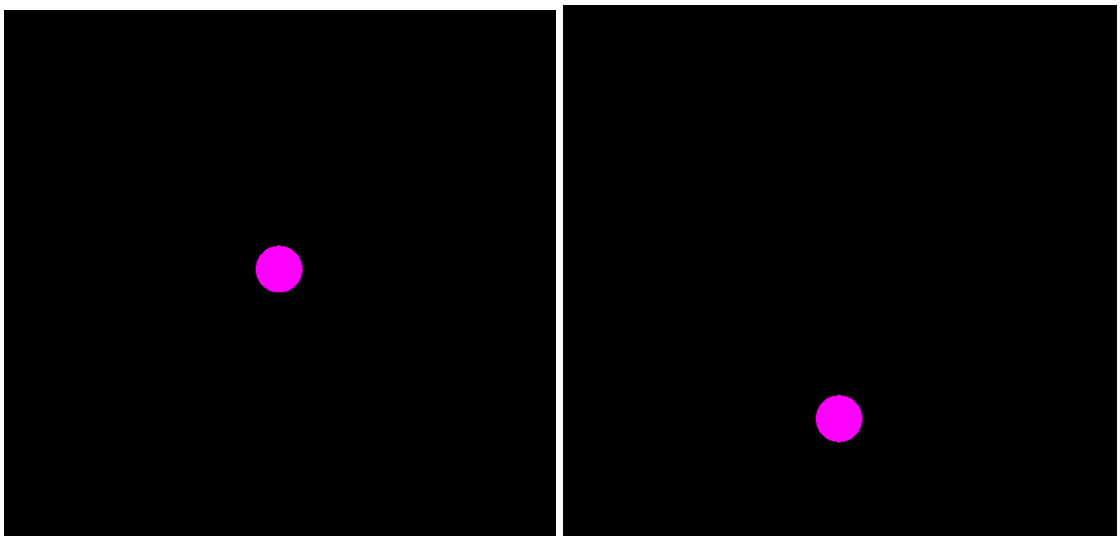
Here is the vert. file:

```
attribute float c;          //get object c
attribute vec3 a;           //set a for rotation
void main(void)
{
   vec4 v4;
   v4 = gl_Vertex;
   mat4 m0 = mat4      //4x4 transformation matrix
   (
         cos (a.x), sin (a.x), 0, 0,
         -sin (a.x), cos (a.x), 0, 0,
         0, 0, 1, 0,
      0, 0, 0, 1
   );
  v4 = m0 * v4;
 gl_Position = gl_ProjectionMatrix *  gl_ModelViewMatrix * v4;
}
```

Here is the frag. file:

```
uniform vec4 color0;
uniform vec4 color1;
uniform vec4 color2;
void main(void)
{
 vec4 color = color0 + color1;
 gl_FragColor = color;          // color
}
```

3. Wrote a shader program that simulates a sphere bouncing.



Here is the source code to the cpp file:

```
GLuint programObject = 0;
GLuint vertexShaderObject = 0;
GLuint fragmentShaderObject = 0;
static GLint win = 0;
int cLoc;
float theta[3];
int  thetaLoc[3];
int  aLoc;        //holds attribute of a
int timeCounter = 0;    // time counter
int time_interval = 10;// how much time has to be before action is done
float x = 0, y = 2.7, z = 0;
float animation_speed = 0.05;
float time_fall = 0.0;
bool can_bounce = false;
float min_height = -2.7;
float gravity = 9.8;
int readShaderSource(char *fileName, GLchar **shader )
{
   FILE *fp;
   int count, pos, shaderSize;
   fp = fopen( fileName, "r");
   if ( !fp ){ return 0;}
   pos = (int) ftell ( fp );
   fseek ( fp, 0, SEEK_END );
   shaderSize = ( int ) ftell ( fp ) - pos;
   fseek ( fp, 0, SEEK_SET );
   if ( shaderSize <= 0 )
   {
      printf("Shader %s empty\n", fileName);
      return 0;
```

```
    }
    *shader = (GLchar *) malloc( shaderSize + 1);
    count = (int) fread(*shader, 1, shaderSize, fp);
    (*shader)[count] = '\0';
    if (ferror(fp)){count = 0;}
    fclose(fp);
    return 1;
}
void checkError()
{
    char *infoLog;
    int  infoLen, len;
    unsigned int shaders[2] = {vertexShaderObject,  fragmentShaderObject};
    char names[][50] = {"Vertex shader", "Fragment shader"};
    for ( int i = 0; i < 2; i++ )
    {
        glGetShaderiv( shaders[i], GL_INFO_LOG_LENGTH, &infoLen);
        if ( infoLen > 0 )
        {
            infoLog = (char *) malloc ( infoLen + 1 );
            glGetShaderInfoLog ( shaders[i], infoLen, &len, infoLog );
            printf("\n***** %s: %s\n", names[i], infoLog );
            delete infoLog;
        }
    }
}
int installShaders(const GLchar *vertex, const GLchar *fragment)
{
    GLint  vertCompiled, fragCompiled;  // status values
    GLint  linked;
    vertexShaderObject = glCreateShader(GL_VERTEX_SHADER);
    fragmentShaderObject = glCreateShader(GL_FRAGMENT_SHADER);
    glShaderSource(vertexShaderObject, 1, &vertex, NULL);
    glShaderSource(fragmentShaderObject, 1, &fragment, NULL);
    glCompileShader(vertexShaderObject);
    glGetShaderiv(vertexShaderObject, GL_COMPILE_STATUS, &vertCompiled);
    glCompileShader( fragmentShaderObject );
    glGetShaderiv( fragmentShaderObject, GL_COMPILE_STATUS, &fragCompiled);
    checkError();
    printf("vertCompiled, fragCompiled: %d, %d\n", vertCompiled, fragCompiled);
    if (!vertCompiled || !fragCompiled){return 0;}
    programObject = glCreateProgram();
    glAttachShader( programObject, vertexShaderObject);
    glAttachShader( programObject, fragmentShaderObject);
    glLinkProgram(programObject);
    glGetProgramiv(programObject, GL_LINK_STATUS, &linked);
```

```
    printf("linked=%d\n");
    if (!linked){return 0;}
    glUseProgram(programObject);
    return 1;
}
int init(void)
{
  const char *version;
  GLchar *VertexShaderSource, *FragmentShaderSource;
  int loadstatus = 0;
  version = (const char *) glGetString(GL_VERSION);
  if (version[0] < '2' || version[1] != '.')
  {
        printf("This program requires OpenGL > 2.x, found %s\n", version);
        exit(1);
  }
  printf("version=%s\n", version );
  readShaderSource((char *) "template.vert", &VertexShaderSource );
  readShaderSource( (char *) "template.frag", &FragmentShaderSource );
  loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);
  char names1[][20] = { "ax", "ay", "az" };
  for ( int i = 0; i < 3; i++ )
  {
        thetaLoc[i] = glGetAttribLocation ( programObject, names1[i] );
        if ( thetaLoc[i] < 0 ) {cout << "No such name: " << names1[i] << endl;}
   }
  aLoc = glGetAttribLocation ( programObject, "a" );          //set a to aloc
  return loadstatus;
}
static void Reshape(int width, int height)       //FOV
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-1.0, 1.0, -1.0, 1.0, 5.0, 25.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0f, 0.0f, -15.0f);
}
void CleanUp(void)    //for closing program
{
  glDeleteShader(vertexShaderObject);
  glDeleteShader(fragmentShaderObject);
  glDeleteProgram(programObject);
  glutDestroyWindow(win);
}
```

```
static void Idle(void)  //this controls timer for animation
{
        float position = y;
        if (timeCounter < time_interval){timeCounter++;}
        else
        {
                timeCounter = 0;
                if (can_bounce == false){y = position - (0.5 * 9.8 * time_fall);//fall down
                }
                else{y = position + 0.75 * (0.5 * 9.8 * time_fall);}    //bounce up
        }
        if (can_bounce == false){time_fall += 0.00025;}
        else{time_fall -= 0.00025;}
        if (y <= min_height){can_bounce = true;}
        if (time_fall < 0){can_bounce = false;}
  glutPostRedisplay();
}
static void Key(unsigned char key, int x, int y)
{
  switch(key)
  {
        case 27:
                CleanUp();      //call clean up function
                exit(0);//close program
                break;
  }
  glutPostRedisplay();
}
void display(void)
{
  GLfloat vec[4];
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glClearColor( 0.0, 0.0, 0.0, 1.0 );    //get black background color
  glColor3f ( 0, 0, 1 );                 //red, this will have no effect if shader is loaded
  cLoc = glGetAttribLocation ( programObject, "c" ); //sets object for .vert shader
  for ( int i = 0; i < 3; i++ ){ glVertexAttrib1f ( thetaLoc[i], theta[i] );}
  glVertexAttrib3f ( aLoc, x, y, z );
  glutSolidSphere(0.25, 50, 50);
  glutSwapBuffers();
  glFlush();
}
int main(int argc, char *argv[])
{
  int success = 0;
  glutInit(&argc, argv);
  glutInitWindowPosition( 0, 0);
```

```
  glutInitWindowSize(600, 600);
  glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
  win = glutCreateWindow("lab2 part1");
  glutReshapeFunc(Reshape);
  glutKeyboardFunc(Key);
  glutDisplayFunc(display);
  glutIdleFunc(Idle);
  glewInit();
  success = init();
  printf("success=%d\n", success );
  if ( success ){glutMainLoop();}
  return 0;
}
```

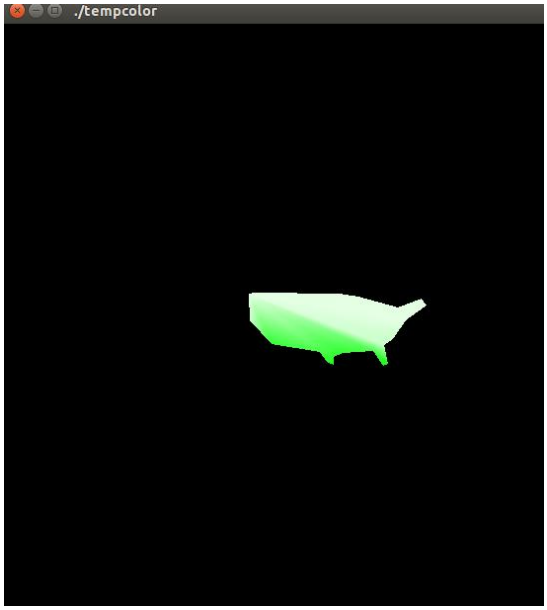
Here is the vert. file:

```
attribute float c;        //get object c
attribute vec3 a;         //set for transformations
void main(void)
{
        vec4 v4;
        v4 = gl_Vertex;
        mat4 m0 = mat4        //4x4 transformation matrix
        (
                1, 0, 0, 0,
                0, 1, 0, 0,
                0, 0, 1, 0,
                0, 0, 0, 1
        );
        v4 = m0 * v4;
        v4.y += a.y;
        gl_Position = gl_ProjectionMatrix *  gl_ModelViewMatrix * v4;
}
```

Here is the frag. file:

```
void main(void){ gl_FragColor = vec4( 1, 0, 1, 1);}
```

4. Wrote a shader program that displays winter temp. in the United States by mixing hot and cold colors together.



Here is the source code to the cpp file:

```
GLuint programObject = 0;
GLuint vertexShaderObject = 0;
GLuint fragmentShaderObject = 0;
static GLint win = 0;
int readShaderSource(char *fileName, GLchar **shader )
{
   FILE *fp;
   int count, pos, shaderSize;
   fp = fopen( fileName, "r");
   if ( !fp ){return 0;}
   pos = (int) ftell ( fp );
   fseek ( fp, 0, SEEK_END );
   shaderSize = ( int ) ftell ( fp ) - pos;
   fseek ( fp, 0, SEEK_SET );
   if ( shaderSize <= 0 )
   {
      printf("Shader %s empty\n", fileName);
      return 0;
   }
   *shader = (GLchar *) malloc( shaderSize);
   if ( *shader == NULL ){printf("memory allocation error\n"); }
   count = (int) fread(*shader, 1, shaderSize, fp);
   (*shader)[count] = '\0';
   if (ferror(fp)){count = 0;}
   fclose(fp);
   return 1;
}
int installShaders(const GLchar *vertex, const GLchar *fragment)
```

```
{
   GLint  vertCompiled, fragCompiled;  // status values
   GLint  linked;
   vertexShaderObject = glCreateShader(GL_VERTEX_SHADER);
   fragmentShaderObject = glCreateShader(GL_FRAGMENT_SHADER);
   glShaderSource(vertexShaderObject, 1, &vertex, NULL);
   glShaderSource(fragmentShaderObject, 1, &fragment, NULL);
   glCompileShader(vertexShaderObject);
   glGetShaderiv(vertexShaderObject, GL_COMPILE_STATUS, &vertCompiled);
   glCompileShader( fragmentShaderObject );
   glGetShaderiv( fragmentShaderObject, GL_COMPILE_STATUS, &fragCompiled);
   printf("vertCompiled, fragCompiled: %d, %d\n", vertCompiled, fragCompiled);
   if (!vertCompiled || !fragCompiled){return 0;}
   programObject = glCreateProgram();
   glAttachShader( programObject, vertexShaderObject);
   glAttachShader( programObject, fragmentShaderObject);
   glLinkProgram(programObject);
   glGetProgramiv(programObject, GL_LINK_STATUS, &linked);
   if (!linked){return 0;}
   glUseProgram(programObject);
   GLchar names[][20] = {
         "CoolestColor", "HottestColor", "CoolestTemp", "TempRange"
   };
   GLint loc[10];
   for ( int i = 0; i < 4; ++i )
   {
         loc[i] =  glGetUniformLocation(programObject, names[i]);
         if (loc[i] == -1) {printf("No such uniform named %s\n", names[i]);}
   }
   glUniform3f(loc[0], 1.0, 1.0, 1.0);          //cold, white
   glUniform3f(loc[1], 0.0, 1.0, 0.0);          //warm, green
   glUniform1f(loc[2], 0.0);
   glUniform1f(loc[3], 1.0);
   return 1;
}
int init(void)
{
  const char *version;
  GLchar *VertexShaderSource, *FragmentShaderSource;
  int loadstatus = 0;
  version = (const char *) glGetString(GL_VERSION);
  if (version[0] != '2' || version[1] != '.')
  {printf("This program requires OpenGL 2.x, found %s\n", version);}
  readShaderSource( (char *) "tempcolor.vert", &VertexShaderSource );
  readShaderSource( (char *) "tempcolor.frag", &FragmentShaderSource );
  loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);
```

```
  return loadstatus;
}
static void Reshape(int width, int height)
{
  glViewport(0, 0, width, height);
  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum(-1.0, 1.0, -1.0, 1.0, 5.0, 25.0);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef(0.0f, 0.0f, -15.0f);
}
void CleanUp(void)
{
  glDeleteShader(vertexShaderObject);
  glDeleteShader(fragmentShaderObject);
  glDeleteProgram(programObject);
  glutDestroyWindow(win);
}
static void Idle(void){glutPostRedisplay();}
static void Key(unsigned char key, int x, int y)
{
  switch(key)
  {
        case 27:
              CleanUp();
              exit(0);
              break;
  }
  glutPostRedisplay();
}
void display(void)
{
  GLfloat vec[4];
  int loc;
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glClearColor( 0.0, 0.0, 0.0, 0.0 );    //black background color
  glColor3f ( 1, 0, 0 );                 //red, this will have no effect if shader is loaded
  loc = glGetAttribLocation(programObject, "VertexTemp" );
  glBegin(GL_POLYGON);
        glVertexAttrib1f(loc, 0.1);
        glVertex3f(-0.5, 0.27, 0.0);
        glVertexAttrib1f(loc, 0.2);
        glVertex3f(-0.49, 0.0, 0.0);
        glVertexAttrib1f(loc, 0.8);
        glVertex3f(-0.27, -0.24, 0.0);
```

```
        glVertexAttrib1f(loc, 0.7);
        glVertex3f(0.22, -0.32, 0.0);
        glVertexAttrib1f(loc, 0.8);
        glVertex3f(0.45, -0.33, 0.0);
        glVertexAttrib1f(loc, 0.7);
        glVertex3f(0.76, -0.31, 0.0);
        glVertexAttrib1f(loc, 0.2);
        glVertex3f(0.87, -0.25, 0.0);
        glVertexAttrib1f(loc, 0.2);
        glVertex3f(0.96, -0.19, 0.0);
        glVertexAttrib1f(loc, 0.1);
        glVertex3f(1.09, 0.0, 0.0);
        glVertexAttrib1f(loc, 0.1);
        glVertex3f(1.01, 0.13, 0.0);
        glVertexAttrib1f(loc, 0.1);
        glVertex3f(0.61, 0.24, 0.0);
        glVertexAttrib1f(loc, 0.1);
        glVertex3f(0.42, 0.27, 0.0);
        glVertexAttrib1f(loc, 0.1);
        glVertex3f(-0.44, 0.28, 0.0);
glEnd();
glBegin(GL_POLYGON);
        glVertexAttrib1f(loc, 0.7);
        glVertex3f(0.22, -0.32, 0.0);
        glVertexAttrib1f(loc, 0.8);
        glVertex3f(0.29, -0.42, 0.0);
        glVertexAttrib1f(loc, 0.9);
        glVertex3f(0.36, -0.45, 0.0);
        glVertexAttrib1f(loc, 0.8);
        glVertex3f(0.36, -0.37, 0.0);
        glVertexAttrib1f(loc, 0.8);
        glVertex3f(0.45, -0.33, 0.0);
glEnd();
glBegin(GL_POLYGON);
        glVertexAttrib1f(loc, 0.7);
        glVertex3f(0.76, -0.31, 0.0);
        glVertexAttrib1f(loc, 0.9);
        glVertex3f(0.86, -0.46, 0.0);
        glVertexAttrib1f(loc, 0.9);
        glVertex3f(0.91, -0.44, 0.0);
        glVertexAttrib1f(loc, 0.2);
        glVertex3f(0.87, -0.25, 0.0);
glEnd();
glBegin(GL_POLYGON);
        glVertexAttrib1f(loc, 0.1);
        glVertex3f(1.09, 0.0, 0.0);
```

```
            glVertex3f(1.3, 0.15, 0.0);
            glVertex3f(1.25, 0.22, 0.0);
            glVertex3f(1.01, 0.13, 0.0);
   glEnd();
   glutSwapBuffers();
   glFlush();
}
int main(int argc, char *argv[])
{
   int success = 0;
   glutInit(&argc, argv);
   glutInitWindowPosition( 0, 0);
   glutInitWindowSize(600, 600);
   glutInitDisplayMode(GLUT_RGB | GLUT_DOUBLE | GLUT_DEPTH);
   win = glutCreateWindow(argv[0]);
   glutReshapeFunc(Reshape);
   glutKeyboardFunc(Key);
   glutDisplayFunc(display);
   glutIdleFunc(Idle);
    glewInit();
   success = init();
   printf("success=%d\n", success );
   if ( success ){glutMainLoop();}
   return 0;
}
```

Here is the vert. file:

```
uniform float CoolestTemp;
uniform float TempRange;
attribute float VertexTemp;
varying float Temperature;
void main(void)
{
  Temperature = ( VertexTemp - CoolestTemp ) / TempRange;
  gl_Position     = ftransform();
}
```

Here is the frag. file:

```
uniform vec3 CoolestColor;
uniform vec3 HottestColor;
varying float Temperature;
void main(void)
{
  vec3 color = mix ( CoolestColor, HottestColor, Temperature );
  gl_FragColor = vec4(color, 1.0);
}
```