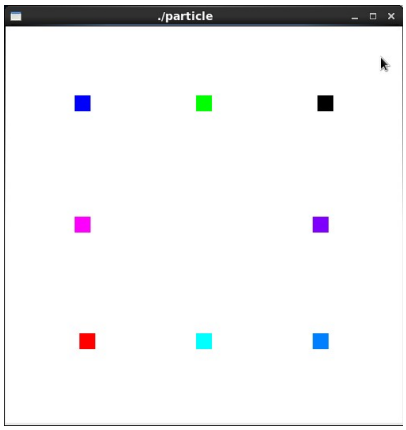
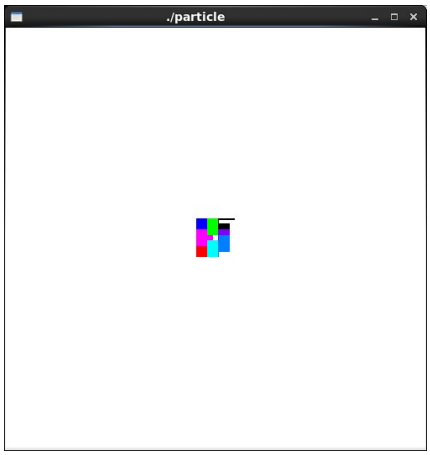
CSE520
Samuel Marrujo
Professor Yu
Lab 08

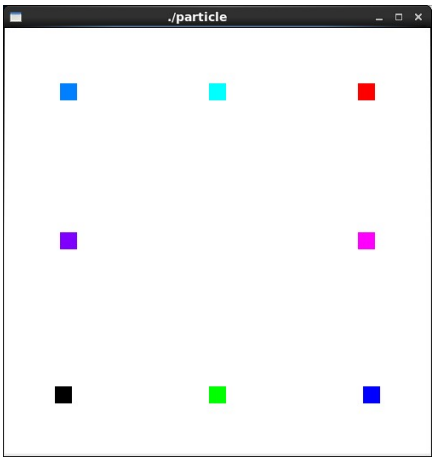
Draw two particles going opposite directions using GLSL

In this lab, we are to create 2 particles with different directions using the vertex and fragment shaders. I have successfully completed this task, and here are pictures of the following:

I decided to put some extra effort and did the following:







```
/*
  particlie.cpp
  Sample program showing how to write GL shader programs.
  Shader sources are in files "particle.vert" and "tests.frag".
  @Author: T.L. Yu, 2009
*/
#include <stdlib.h>
#include <stdio.h>
#include <string.h>
#include <fcntl.h>
#include <sys/types.h>
#include <unistd.h>

#define GLEW_STATIC 1
#include <GL/glew.h>
#include <GL/glu.h>
#include <GL/glut.h>


using namespace std;

/*
   Global handles for the currently active program object, with its two shader
objects
*/
GLuint programObject = 0;
GLuint vertexShaderObject = 0;
GLuint fragmentShaderObject = 0;
static GLint win = 0;

GLuint timeParam;       //parameters for sending to vertex shader
GLuint velParam;        //parameters for sending to vertex shader

int readShaderSource(char *fileName, GLchar **shader )
{
    // Allocate memory to hold the source of our shaders.
    FILE *fp;
    int count, pos, shaderSize;

    fp = fopen( fileName, "r");
    if ( !fp )
        return 0;

    pos = (int) ftell ( fp );
    fseek ( fp, 0, SEEK_END );                  //move to end
    shaderSize = ( int ) ftell ( fp ) - pos;    //calculates file size
    fseek ( fp, 0, SEEK_SET );                  //rewind to beginning

    if ( shaderSize <= 0 ){
        printf("Shader %s empty\n", fileName);
        return 0;
    }

    *shader = (GLchar *) malloc( shaderSize + 1);

    // Read the source code

    count = (int) fread(*shader, 1, shaderSize, fp);
```

```
    (*shader)[count] = '\0';

    if (ferror(fp))
        count = 0;

    fclose(fp);

    return 1;
}


//  public
int installShaders(const GLchar *vertex, const GLchar *fragment)
{
    GLint  vertCompiled, fragCompiled;  // status values
    GLint  linked;

    // Create a vertex shader object and a fragment shader object

    vertexShaderObject = glCreateShader(GL_VERTEX_SHADER);
    fragmentShaderObject = glCreateShader(GL_FRAGMENT_SHADER);

    // Load source code strings into shaders, compile and link

    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;

    // Create a program object and attach the two compiled shaders

    programObject = glCreateProgram();
    glAttachShader( programObject, vertexShaderObject);
    glAttachShader( programObject, fragmentShaderObject);

    // Link the program object

    glLinkProgram(programObject);
    glGetProgramiv(programObject, GL_LINK_STATUS, &linked);

    if (!linked)
        return 0;

    // Install program object as part of current state

    glUseProgram(programObject);

    //check log
    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);
   if (version[0] != '2' || version[1] != '.') {
      printf("This program requires OpenGL 2.x, found %s\n", version);
//      exit(1);
   }
   readShaderSource( (char *) "particle.vert", &VertexShaderSource );
   readShaderSource((char *) "tests.frag", &FragmentShaderSource );
   loadstatus = installShaders(VertexShaderSource, FragmentShaderSource);

   timeParam = glGetUniformLocation ( programObject, "time" );
   velParam = glGetAttribLocation ( programObject, "vel" );
   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)
{
   float t = glutGet ( GLUT_ELAPSED_TIME );
   while ( t > 1800 ) t -= 1800;
   glUniform1f( timeParam, t );
   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 = glGetAttribLocation(programObject, "temp");
   glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
   glClearColor(1.0, 1.0, 1.0, 0.0);            //get white background color
   glColor3f(1, 0, 0);                    //red, this will have no effect if
shader is loaded
   glPointSize(20);
   //"shoot" a particle at 45 degrees
   glBegin (GL_POINTS);  //need "GL_POINTS"; "GL_POINT" does not work
     glVertexAttrib3f(velParam, 20, 30, 0 );    //send vel to vertex shader
     glVertexAttrib3f(loc, 1, 0, 0 );
     glVertex2f (-15, -15);                //starting position of particle
   glEnd();
   glBegin(GL_POINTS);
     glVertexAttrib3f(velParam, 20, -10, 0);
     glVertexAttrib3f(loc, 0, 0, 1);
     glVertex2f(-15, 15);
   glEnd();
   glBegin(GL_POINTS);
     glVertexAttrib3f(velParam, 20, 10, 0);
     glVertexAttrib3f(loc, 1, 0, 1);
     glVertex2f(-15, 0);
   glEnd();
   glBegin(GL_POINTS);
     glVertexAttrib3f(velParam, 0, 30, 0);
     glVertexAttrib3f(loc, 0, 1, 1);
     glVertex2f(0, -15);
   glEnd();
   glBegin(GL_POINTS);
     glVertexAttrib3f(velParam, 0, -10, 0);
     glVertexAttrib3f(loc, 0, 1, 0);
     glVertex2f(0, 15);
   glEnd();
   glBegin(GL_POINTS);
     glVertexAttrib3f(velParam, -20, -10, 0);
     glVertexAttrib3f(loc, 0, 0, 0);
     glVertex2f(15, 15);
   glEnd();
   glBegin(GL_POINTS);
     glVertexAttrib3f(velParam, -20, 10, 0);
     glVertexAttrib3f(loc, 0.5, 0, 1);
     glVertex2f(15, 0);
   glEnd();
   glBegin(GL_POINTS);
     glVertexAttrib3f(velParam, -20, 30, 0);
     glVertexAttrib3f(loc, 0, 0.5, 1);
```

```
      glVertex2f(15, -15);
    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);

    // Initialize the "OpenGL Extension Wrangler" library
     glewInit();

    success = init();
    if ( success )
      glutMainLoop();
    return 0;
}
```

vertex shader:

```
//particle.vert

uniform float time;                 //value provided by application program
attribute vec3 vel;                 //value provided by application program
attribute vec3 temp;
varying vec3 color;

void main(void)
{
  color = temp;
  float s = 1000.0;                 //scale factor
  float g = -10.0;
  float t;
  t =  time / s;              //time in ms
  vec4 object_pos = gl_Vertex;            //starting position
  vec4 object2_pos = gl_Vertex;

  object_pos.x = object_pos.x + vel.x*t;
  object_pos.y = object_pos.y + vel.y*t  + g/2*t*t;
  object_pos.z = object_pos.z + vel.z*t;


  gl_Position = gl_ModelViewProjectionMatrix * object_pos;

}
```

frag shader:

```
//tests.frag
//a minimal fragment shader
varying vec3 color;

void main(void)
{
  gl_FragColor = vec4( color, 1 );
  //gl_FragColor = vec3( 0, 0, 1);  // vec3 no longer works
  //gl_FragColor = vec3( 1, 0, 0);  //red color
}
```