Project 1 Discussion

Tom Kelliher, CS 320

Feb. 22, 2013

1 Administrivia

Announcements

Collect Assignment 3.

Assignment

Read 3.1, Appendices B and C.

From Last Time

OpenGL object management and viewing basics, using `collision.cpp`.

Outline

1. Project 1 discussion.

Coming Up

Vector and matrix operations, using `collision.cpp`.

2 Project 1

2.1 World/Window Coordinates

1. Let cell size be 1.0×1.0.

2. Use a one cell wide border around the maze. Special encoding to indicate these cells? Don't render border.

   Orthogonal projection; viewing volume.

3. Use a 500×500 window.

4. Deal with reshape events, but no need to maintain the aspect ratio.

   `reshape()` will record the new width and height, issue a viewport call to use the entire window, and post a redisplay event.

   Actual window width and height needed for pick selection calculation in the mouse callback.

2.2 Cell Data Structure

1. Maze is 2-D array of cells.

2. Needed for each cell: visited; north, south, east, west walls up.

   Redundancy: remove.

   Track/draw south, west maze walls.

3. `struct` declaration:

```
typedef struct cell
{
   int visited;
   int north;
   int east;
} cell;
```

   What about the south and west edges of the maze???

2.3 Sketch of `main()`

```
cell maze[ROWS + 2][COLS + 2];

/* If the south and west edges of the maze are closed with single
 * line segments, only 2 * (ROWS * COLS + 3) elements are needed for
 * each array.
 */
vec2 points[2 * (ROWS + 1) * (COLS + 1)];
vec3 colors[2 * (ROWS + 1) * (COLS + 1)];

main(...)
{
   seed the random number generator;

   set-up glut and the window;
   set-up glew;

   register callbacks for display, reshape, and mouse;

   call init;

   render the maze;

   enter the glut main loop;
}
```

Seeding the random number generator:

```
#include <time.h>

...

   srand((unsigned) time(NULL));
```

2.4 `visit()` Pseudo-Code

Initial call:

```
mark each cell visited/not visited as appropriate and raise all walls;
visit(cell in the lower left corner);
```

```
visit(cell)
{
   mark cell visited;

   Randomly arrange the four compass directions
   (See Fisher-Yates Shuffle in Wikipedia)

   for each of the four randomly arranged compass directions
      if adjacent_cell exists and is unvisited
      {
         remove appropriate wall;
         visit(adjacent_cell);
      }
}
```

2.5 `path()` Pseudo-Code

Idea: Get a sequence of cells from start cell to end cell on the call stack and draw path from end to start.

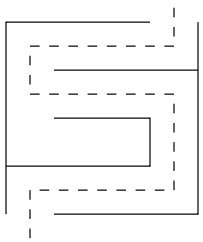
Don't forget to mark all maze cells as unvisited before first call!

Initial call: `path(start_cell, end_cell);`

```
path(current_cell, target_cell)
{
   mark current_cell visited;

   if current cell == target cell
   {
      store vertex for current_cell;
      return 1;
   }

   for each of the four neighbor cells, new_cell
      if new_cell is unvisited
      and there is no wall between current_cell and new_cell
      and (path(new_cell, target_cell)
         {
            store vertex for current_cell;
            return 1;
         }

   return 0;
}
```

2.6 A Sample Maze

4×4 maze:



2.7 Vertex Shader

```
#version 150

in  vec4 vPosition;
in  vec4 vColor;
out vec4 color;

uniform mat4 projection;

void main()
{
    gl_Position = projection * vPosition;
    color = vColor;
}
```

2.8 Fragment Shader

```
#version 150

in  vec4 color;
out vec4 fColor;

void main()
{
    fColor = color;
}
```

2.9 init()

Global scalars are all of type GLuint.

```
void init(void)
{
   glClearColor(1.0, 1.0, 1.0, 0.0);

   program = InitShader("vshader.glsl", "fshader.glsl");
   glUseProgram(program);
   projection = glGetUniformLocation(program, "projection");

   glGenVertexArrays(1, &mazeVAO);
   glBindVertexArray(mazeVAO);

   glGenBuffers(1, &mazeBuffer);
   glBindBuffer(GL_ARRAY_BUFFER, mazeBuffer);
   glBufferData(GL_ARRAY_BUFFER, sizeof(points) + sizeof(colors),
                NULL, GL_DYNAMIC_DRAW);

   GLuint vPosition = glGetAttribLocation(program, "vPosition");
   glEnableVertexAttribArray(vPosition);
   glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE, 0,
                         BUFFER_OFFSET(0));

   GLuint vColor = glGetAttribLocation(program, "vColor");
   glEnableVertexAttribArray(vColor);
   glVertexAttribPointer(vColor, 3, GL_FLOAT, GL_FALSE, 0,
                         BUFFER_OFFSET(sizeof(points)));

   glGenVertexArrays(1, &pathVAO);
   glBindVertexArray(pathVAO);

   glGenBuffers(1, &pathBuffer);
   glBindBuffer(GL_ARRAY_BUFFER, pathBuffer);
   glBufferData(GL_ARRAY_BUFFER, sizeof(points) + sizeof(colors),
                NULL, GL_DYNAMIC_DRAW);

   vPosition = glGetAttribLocation(program, "vPosition");
   glEnableVertexAttribArray(vPosition);
   glVertexAttribPointer(vPosition, 2, GL_FLOAT, GL_FALSE, 0,
                         BUFFER_OFFSET(0));
```

```
   vColor = glGetAttribLocation(program, "vColor");
   glEnableVertexAttribArray(vColor);
   glVertexAttribPointer(vColor, 3, GL_FLOAT, GL_FALSE, 0,
                         BUFFER_OFFSET(sizeof(points)));
}
```

2.10 display()

```
void display(void)
{
   glClear(GL_COLOR_BUFFER_BIT);

   p = Ortho(-1.0, COLS + 1.0, -1.0, ROWS + 1.0, -1.0, 1.0);
   glUniformMatrix4fv(projection, 1, GL_TRUE, p);

   glBindVertexArray(mazeVAO);
   glDrawArrays(GL_LINES, 0, mazeIndex);

   if (endpointCnt == 2)
   {
      glBindVertexArray(pathVAO);
      glDrawArrays(GL_LINE_STRIP, 0, pathIndex);
   }

   glFlush();
}
```

2.11 renderMaze()

```
void renderMaze(void)
{
   /* Create the maze. */

   /* Store vertex data for the maze in points and colors arrays.
    * Use mazeIndex as the number of vertices.
    */

   glBindBuffer(GL_ARRAY_BUFFER, mazeBuffer);
   glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(points), points);
   glBufferSubData(GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors);
```

```
   glutPostRedisplay();
}
```

2.12 renderPath()

```
void renderPath(void)
{
   /* Find a path through the maze, connecting the defined points. */

   /* Store vertex data for the path in points and colors arrays.
    * Use pathIndex as the number of vertices.
    */

   glBindBuffer(GL_ARRAY_BUFFER, pathBuffer);
   glBufferSubData(GL_ARRAY_BUFFER, 0, sizeof(points), points);
   glBufferSubData(GL_ARRAY_BUFFER, sizeof(points), sizeof(colors), colors);

   glutPostRedisplay();
}
```