CSCI 420
COMPUTER GRAPHICS

HOMEWORK 1

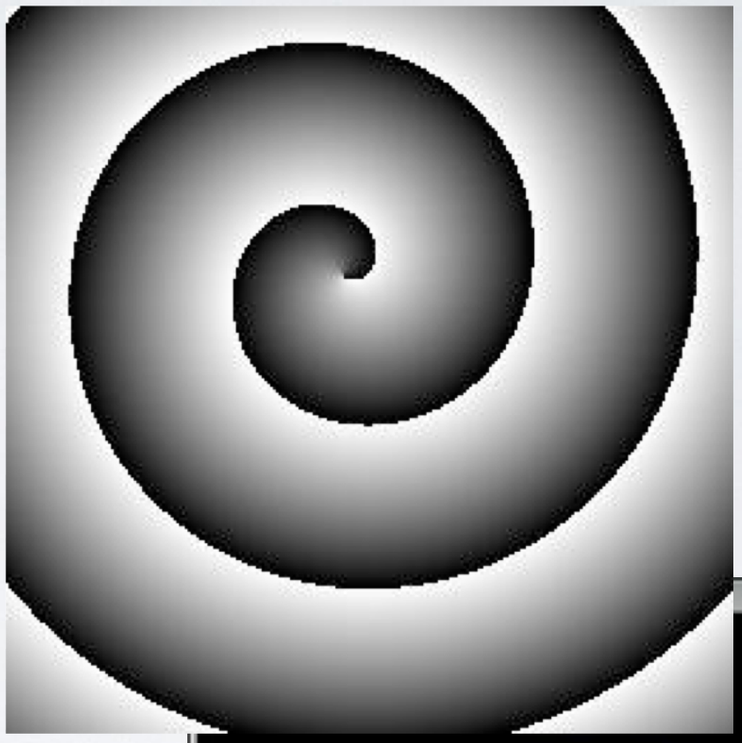
Height Fields

GOALS

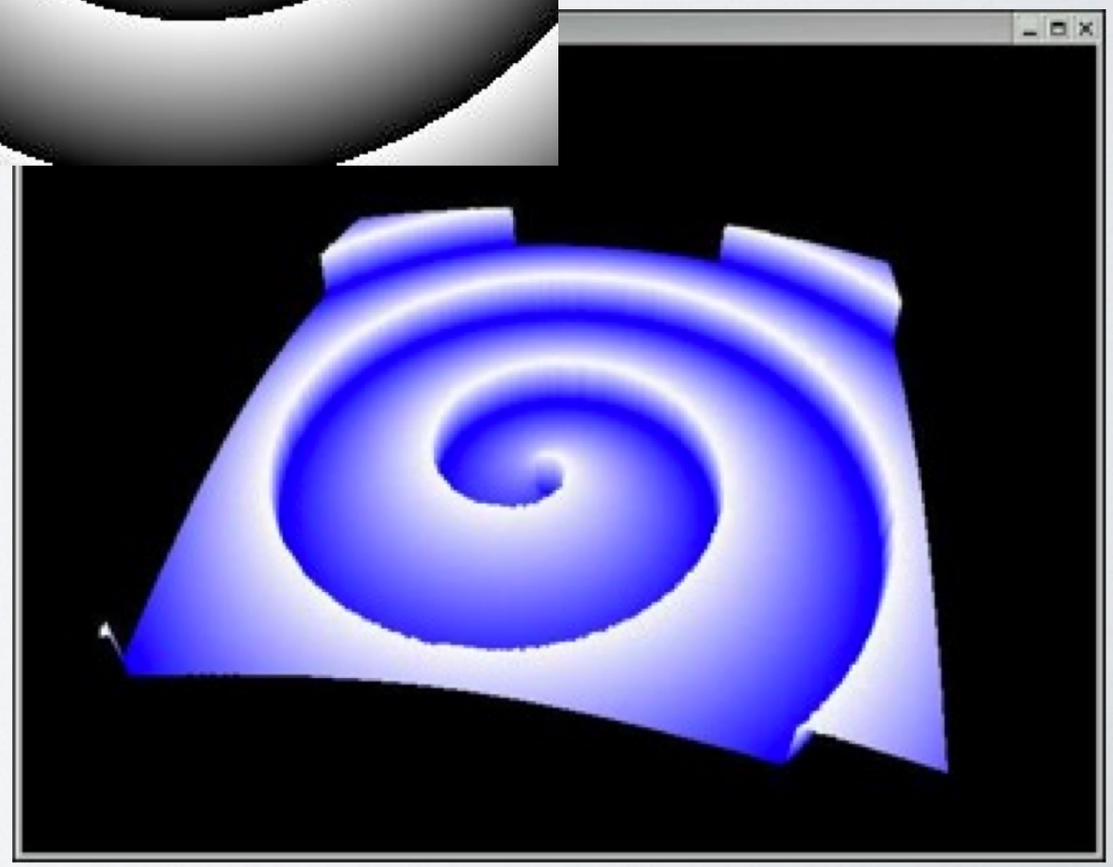
- Refresher on C/C++
- Hands-on introduction to OpenGL and supporting libraries
- Understand Height Fields and Camera manipulation

HEIGHT FIELD

A height field is a visual representation of a function which takes as input a two-dimensional point and returns a scalar value ("height") as output. In other words, a function f takes x and y coordinates and returns a z coordinate.

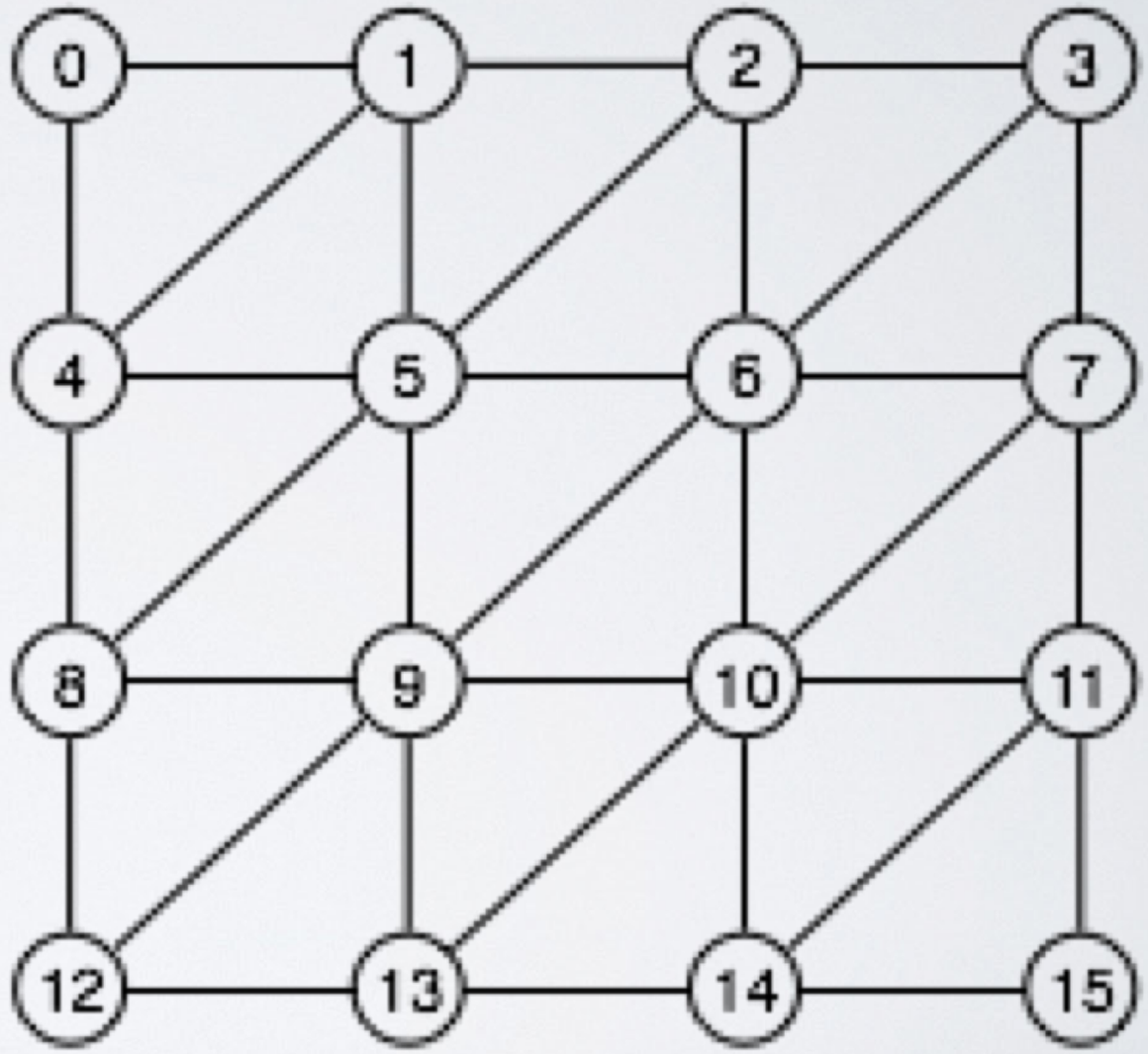


Rendering a height field over arbitrary coordinates is somewhat tricky--we will simplify the problem by making our function piece-wise. Visually, the domain of our function is a two-dimensional grid of points, and a height value is defined at each point. We can render this data using only a point at each defined value, or use it to approximate a surface by connecting the points with triangles in 3D.



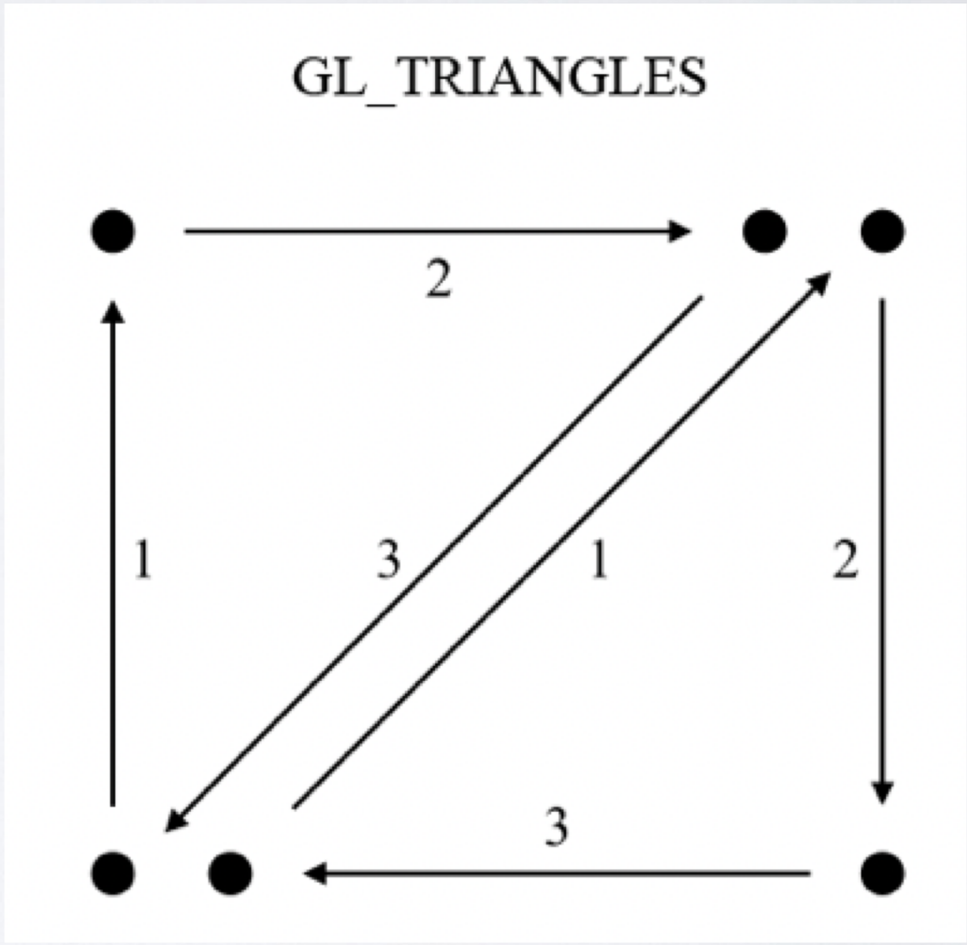
HEIGHT FIELD

- Create a height field based on an input image
- Height field is a function f(x, y) = z, where the pixel at (x, y) in the input image defines the height z at that point
- (x, y, z) defines a vertex in the field
- Render the vertices in OpenGL using triangles



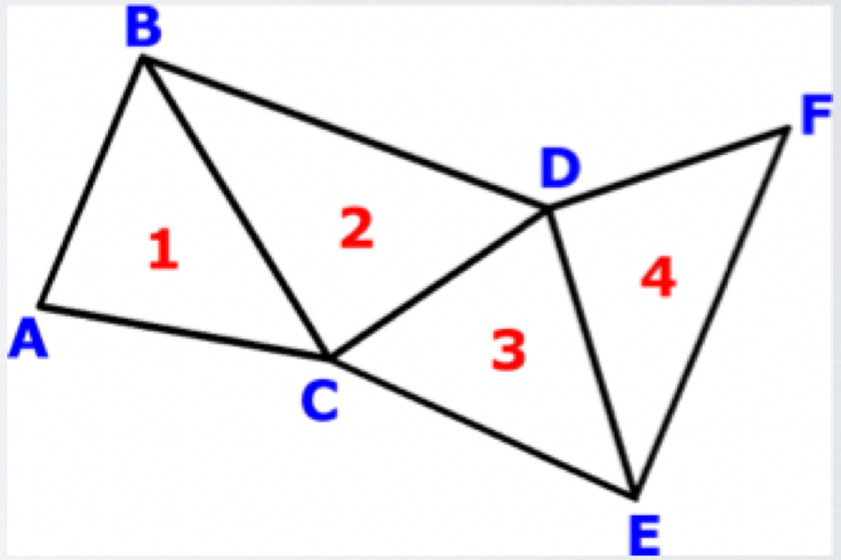
RENDERING TRIANGLES

- Specify each triangle individually as a triplet of vertices
- Requires 3n vertices for n triangles
- Use GL_TRIANGLES



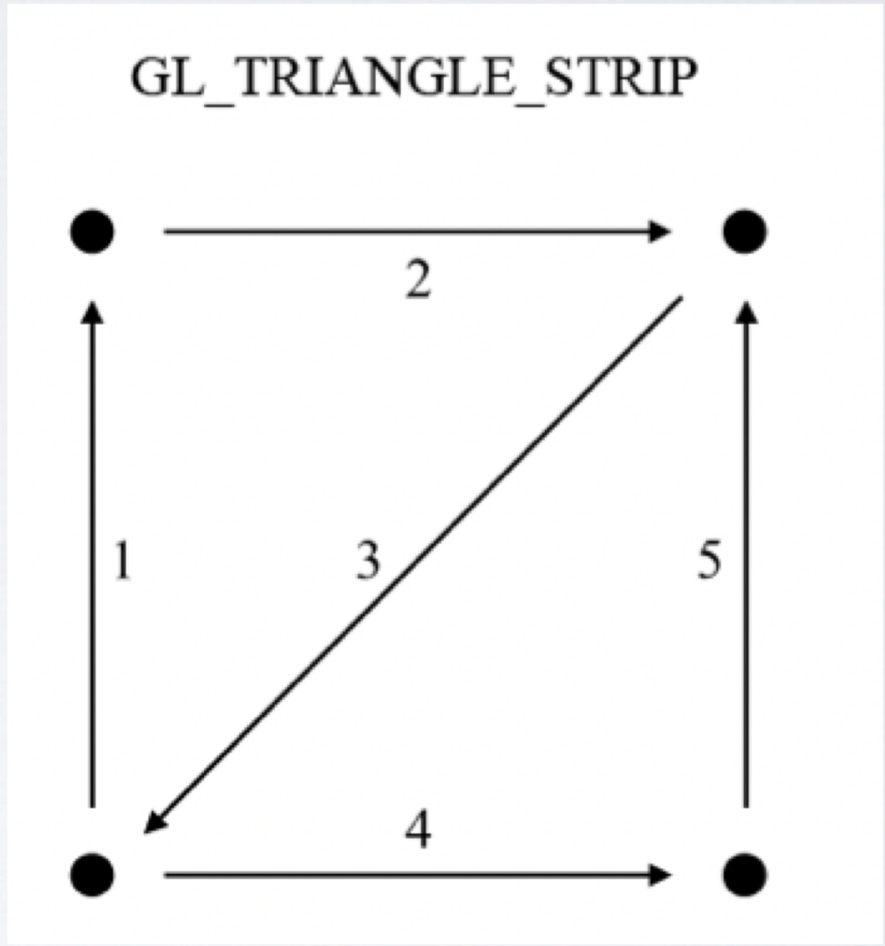
RENDERING TRIANGLES

```
glBegin(GL_TRIANGLES); // specify type of shapes to draw

// draw first triangle
glColor3f(A_Red, A_Green, A_Blue); // first vert color

glVertex3f(A_X, A_Y, A_Z); // first vert pos

glColor3f(B_Red, B_Green, B_Blue);

glVertex3f(B_X, B_Y, B_Z);

glColor3f(C_Red, C_Green, C_Blue);

glVertex3f(C_X, C_Y, C_Z);

// draw second triangle

glColor3f(C_Red, C_Green, C_Blue);

glVertex3f(C_X, C_Y, C_Z);

glColor3f(B_Red, B_Green, B_Blue);

glVertex3f(B_X, B_Y, B_Z);

glColor3f(D_Red, D_Green, D_Blue);

glVertex3f(D_X, D_Y, D_Z);

...
glEnd(); // finish drawing
```



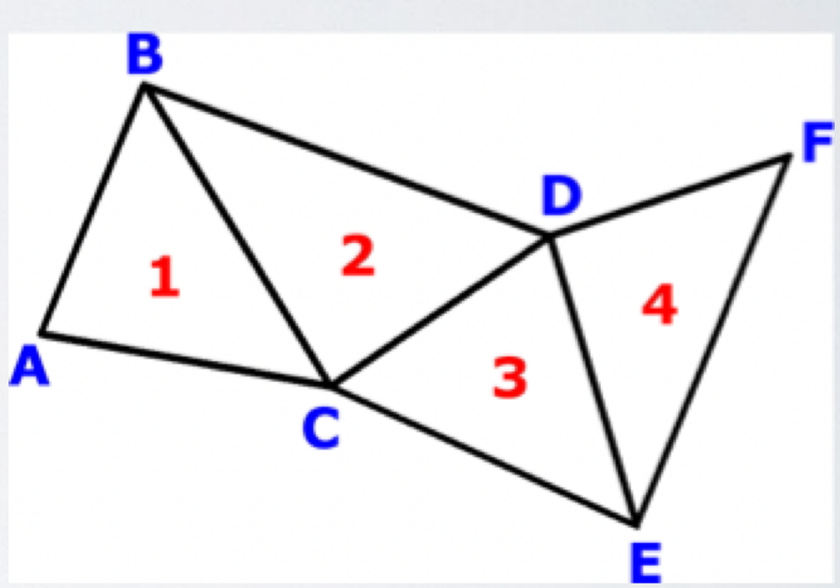
RENDERING TRIANGLES OPTIMIZED

- Render triangles as a strip
- Triangles must be adjacent - sharing an edge
- For each triangle beyond the first, specify only one additional vertex
- 2 + n vertices for n triangles
- Use GL_TRIANGLE_STRIP



RENDERING TRIANGLES OPTIMIZED

```
glBegin(GL_TRIANGLE_STRIP); // specify type of shapes to draw

// specify initial 2 vertices
glColor3f(A_Red, A_Green, A_Blue); // first vert color

glVertex3f(A_X, A_Y, A_Z); // first vert pos

glColor3f(B_Red, B_Green, B_Blue);

glVertex3f(B_X, B_Y, B_Z);

// specify last vertex for first triangle

glColor3f(C_Red, C_Green, C_Blue);

glVertex3f(C_X, C_Y, C_Z);
// specify new vertex for second triangle

glColor3f(D_Red, D_Green, D_Blue);

glVertex3f(D_X, D_Y, D_Z);

// specify new vertex for third triangle

glColor3f(E_Red, E_Green, E_Blue);

glVertex3f(E_X, E_Y, E_Z);

...
glEnd(); // finish drawing
```



LINES OF TRIANGLE STRIPS

Turn pairs of pixel rows ("scanlines") into triangle strips.

```
for(int i = 0; i < pic->ny - 1; i++)
{
  OGL_initialize tri-strip creation

  for(int j = 0; j < pic->nx; j++)
  {
    indx0 = (j, i, z from PIC_PIXEL()) // 'top' vertex
    indx1 = (j, i+1, z from PIC_PIXEL()) // 'bottom' vertex

    // sequential top,bottom vert pairs generates a tri-strip
    OGL_specify() vertex with z=indx0
    OGL_specify() vertex with z=indx1

  }// next pixel in current row

  OGL_end current tri-strip

}// next row
```

RENDERING MODES

- Use glPolygonMode() to control render mode.
  glPolygonMode(GL_FRONT_AND_BACK, mode)
- GL_POINT - Render vertices as points
- GL_LINE - Wireframe
- GL_FILL - Solid triangles

HEIGHT VALUES AND PIXELS

- Heights are specified as grayscale, 8 bits/pixel
- Each height value is an unsigned char/byte (0 - 255)
- Pixel values are held in the ‘pix’ array in the ‘Pic’ data structure

```
typedef struct
{
  int nx, ny;
  int bpp;
  Pixel1 *pix; /* array of pixels*/
}
```

HEIGHT VALUES AND PIXELS

- Consider the following 4x4 (16 pixel) image

```
100 110 120 130
200 215 230 245
250 200 150 100
0    30  60  90
```

- Would be laid in ‘pix’ in “row major” order

```
100 110 120 130 200 215 230 245 250 200 150 100 0 30 60 90
```

ACCESSING PIXEL VALUES

- To access a pixel value at any (x, y), use the macro `PIC_PIXEL(pic, x, y, chan)`

…defined in pic.h, with chan=0.

```
for(int i=0;i<pic->ny;i++)
{
  for(int j=0;j<pic->nx;j++)
  {
    // chan=0, since we're accessing the first/only channel
    unsigned char heightVal = PIC_PIXEL(pix,j,i,0);

    // use heightVal…

  }// next pixel in current row
}// next row
```

CREATING FILENAMES

```
char myFilenm[2048];
for (int i=0;i<1000;i++)
{
  sprintf(myFilenm, "anim.%04d.jpg", i);
  // myFilenm will be anim.0001.jpg,… anim.0999.jpg
  // ..
}
```

TIPS

- Use depth buffer to remove occluded objects.

  `glEnable(GL_DEPTH_TEST)`

  …during initialization

- Clear color and depth buffer before rendering each frame.

  `glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT)`

- Tips on optimizing triangle strip rendering:
  http://dan.lecocq.us/wordpress/2009/12/25/triangle-strip-for-grids-a-construction/

- Guide for catching common C errors. Particularly helpful for those whose "first language" isn't C/C++:
  http://www.drpaulcarter.com/cs/common-c-errors.php

THANKS!