Computer Graphics
(Basic OpenGL)

Thilo Kielmann
Fall 2004
Vrije Universiteit, Amsterdam
kielmann@cs.vu.nl

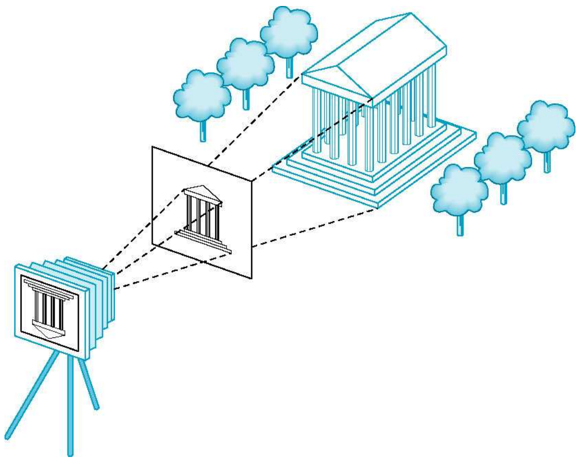
http://www.cs.vu.nl/~graphics/

Outline for today

- OpenGL API and libraries 1×1
- Graphics primitives, attributes, colors
- Simple (orthographic) viewing
- Control and the window system
- 3D graphics

$\Rightarrow$ **OpenGL API and libraries** 1×1

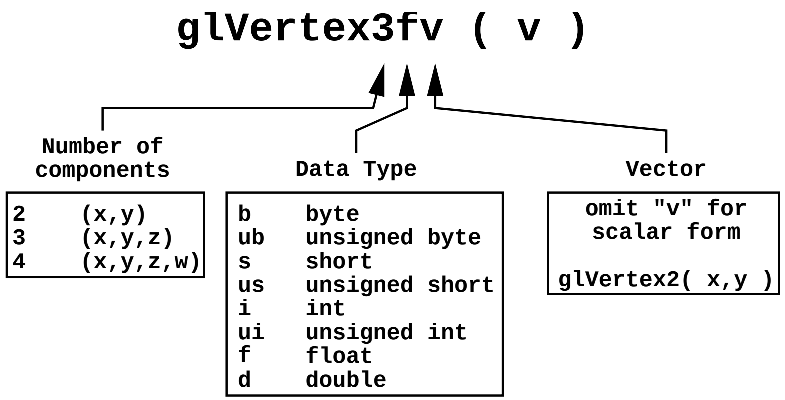
OpenGL: Synthetic Camera



- scene objects
- camera
- projection plane (screen)

OpenGL Describes Objects: Vertices

- a point is called a **vertex**
  - ⋆ user coordinates: possibly infinite drawing pad
- **vertices** (plural of vertex) are always 3D
  - ⋆ can also be used as 2D
- general form: `glVertex*` examples:
  - ⋆ glVertex2i(GLint x, GLint y)
  - ⋆ glVertex3f(GLfloat x, GLfloat y, GLfloat z)
  - ⋆ glVertex3fv(GLfloat[] vertex)



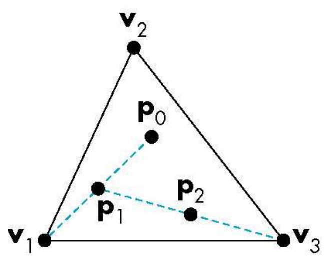
All OpenGL calls follow this general structure.

Vertices are used to build other primitives

```
glBegin(GL_POINTS);
    glVertex2f(x1,y1);
    glVertex2f(x2,y2);
glEnd();
```

```
glBegin(GL_LINES);
    glVertex2f(x1,y1);
    glVertex2f(x2,y2);
glEnd();
```

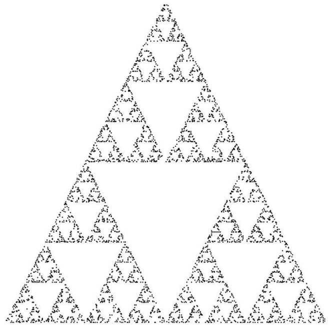

Example: The Sierpinksi Gasket



given $v_1, v_2,$ and v_3
pick p_0 at random
pick one of v_1, v_2, v_3 at random
$p_1 =$ "halfway" between p_0 and vertex
display p_1
replace p_0 by p_1 and continue

Plotting Sierpinski Points

```
void display( void ){
  typedef GLfloat point2[2];
  point2 vertices[3]={{0.0,0.0},{250.0,500.0},{500.0,0.0}};
  point2 p ={75.0,50.0}; /* initial point inside triangle */
  int j, k, rand();
  for ( k=0; k<5000; k++) {
     j=rand() %3; /* pick a vertex at random */
     p[0] = (p[0]+vertices[j][0])/2.0;
     p[1] = (p[1]+vertices[j][1])/2.0;
     glBegin(GL_POINTS);
        glVertex2fv(p);
     glEnd();
  }
  glFlush(); /* clear buffers */
}
```



Still Open Questions:

1. In what colors are we drawing?
2. Where on the screen does our image appear?
3. How large will the image be?
4. How do we create a window for the image?
5. How much of our infinite pad will appear on the screen?
6. How long will the image remain on the screen?

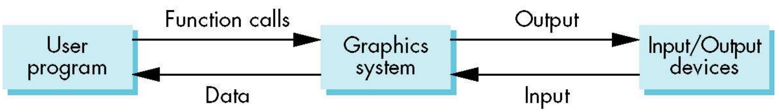
Answering these Questions: Categories of Graphics Functions

1. primitive functions (objects: "what")
2. attribute functions ("how")
3. viewing functions (camera)
4. transformation functions (e.g., rotation . . .)
5. input functions
6. control functions

The Graphics State Machine

First, attribute functions set how vertices will be displayed.

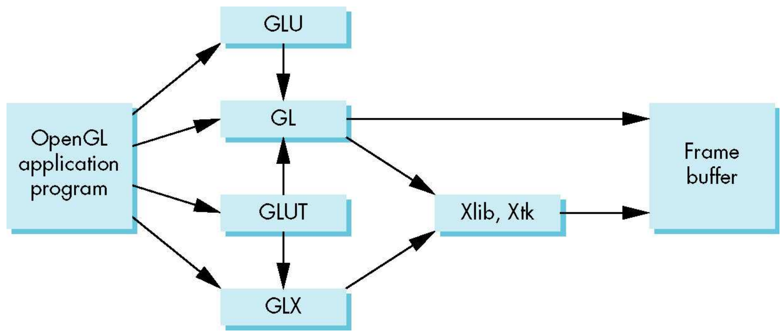
Then, vertices are drawn, according to the current state. (According to all previous calls to the attribute functions.)



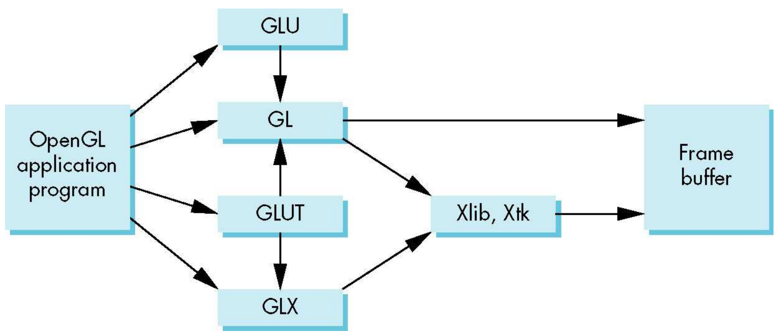
OpenGL Library Structure

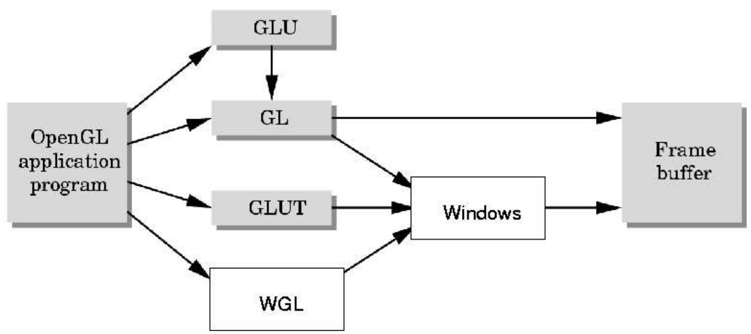
```
#include <GL/glut.h>      or:
#include <glut.h>
```

```
glFunction()
gluFunction()
glutFunction()
```



OpenGL Library Structure (Unix vs. Windows)

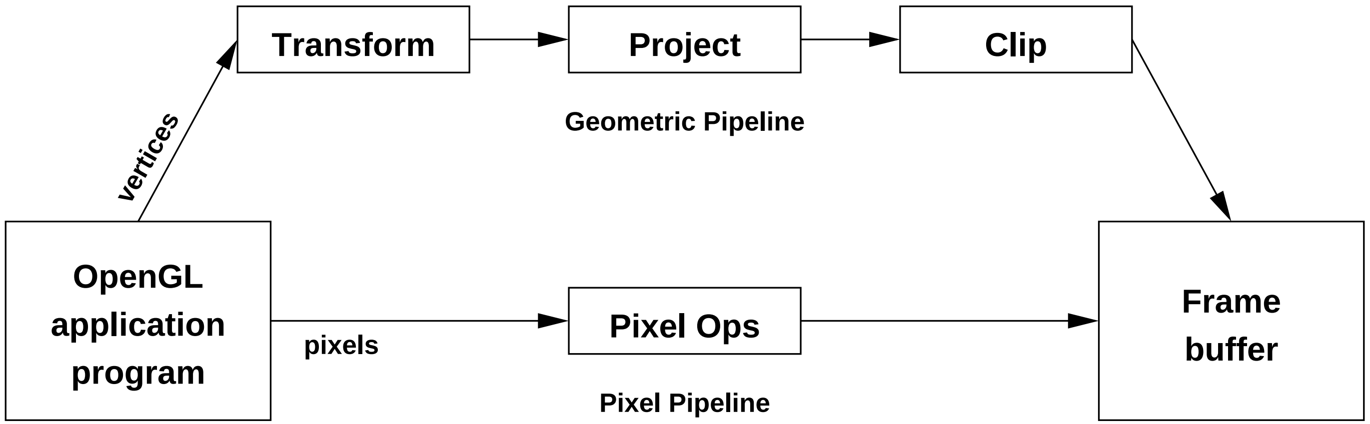




Only use `GL`, `GLU`, and `GLUT` calls for portable programs!

(Check the documentation for our own header file.)

The OpenGL (double) Pipeline



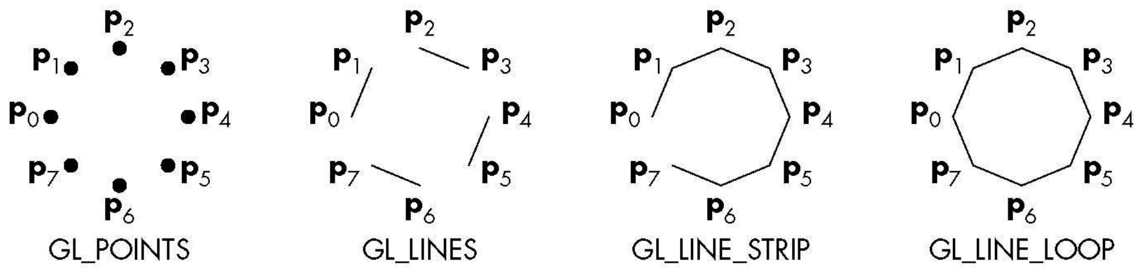
Outline for today

- OpenGL API and libraries 1×1
- Graphics primitives, attributes, colors
- Simple (orthographic) viewing
- Control and the window system
- 3D graphics

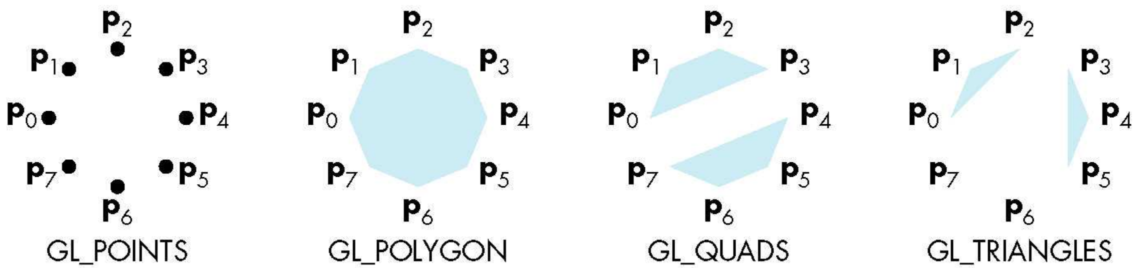
$\Rightarrow$ **Graphics primitives, attributes, colors**

Geometric Primitive Elements

```
glBegin( ... );
  glVertex*( ... );
  .
  .
glEnd();
```



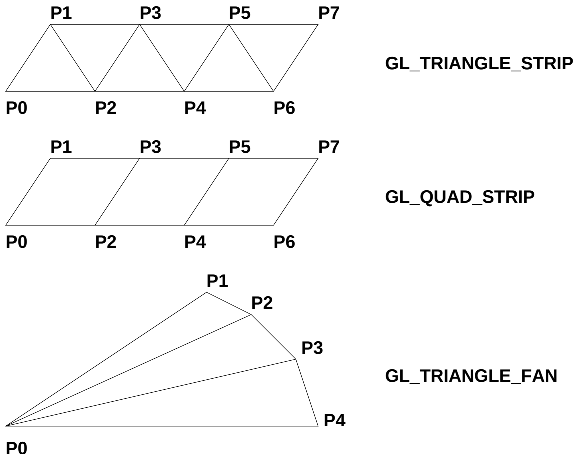
Polygon Types



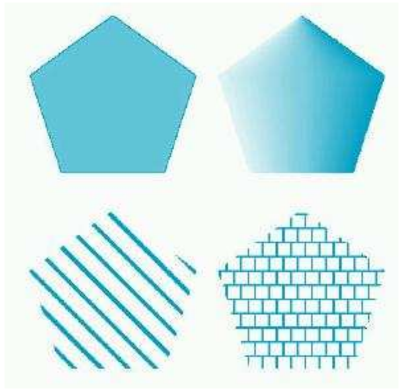
The appearance of polygons depends on the attributes that have been set before.

(This is the same as with lines.)

Polygon Strips

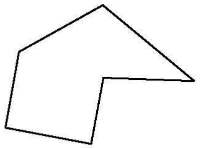


Polygons can be Filled

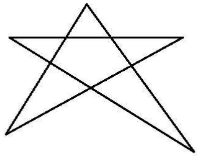


Filling the Polygon Interior (2D)

To be filled, polygons have to be: **simple** and **convex**.



(a)

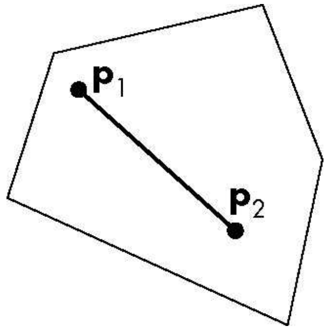


(b)

A *simple* polygon has a well-defined interior.

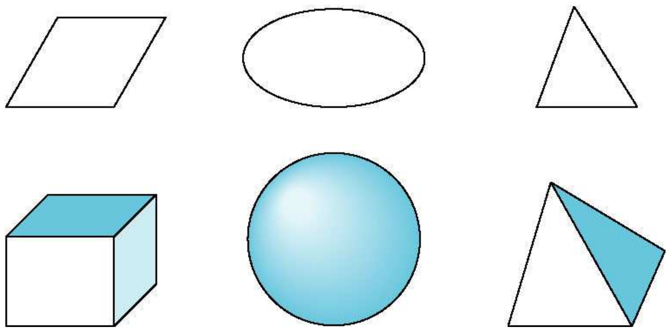
(a) simple

(b) non-simple



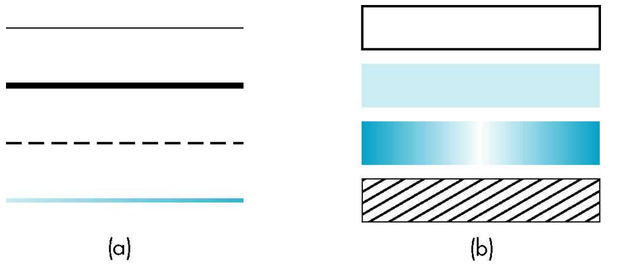
“All points on the line segment between any 2 points inside the polygon are inside the polygon.”

Filling Polygons (3D)



Polygons have to be simple, convex, and **flat**. This often boils down to triangles!

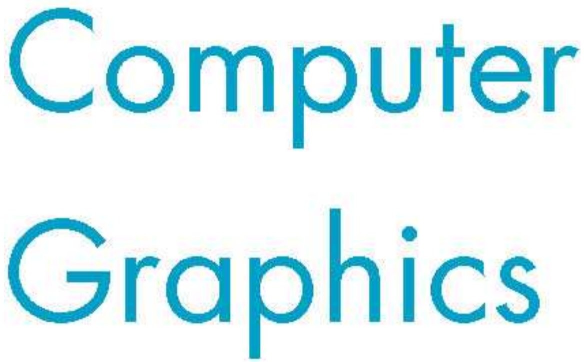
Attributes for Lines and Polygons

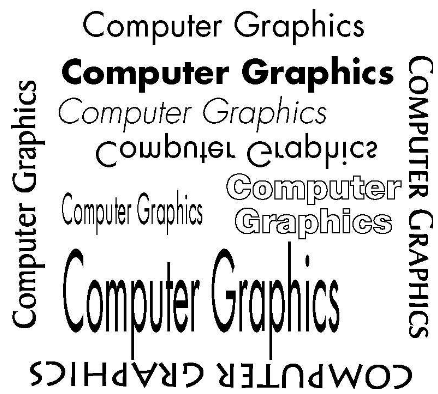


Text (Raster Text)

Raster text goes into the pixel pipeline.

Text (Stroke Text)





Stroke text can be treated like all other graphics objects.

Fonts in GLUT

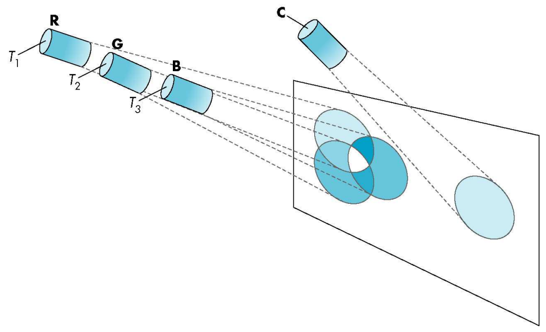
Stroke fonts: (have to be scaled)

```
glutStrokeCharacter( GLUT_STROKE_MONO_ROMAN, int k)
glutStrokeCharacter( GLUT_STROKE_ROMAN, int k)
```

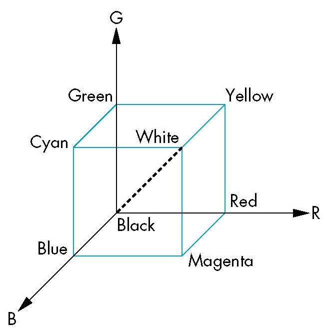
Bitmap fonts: (written into the pixel pipeline)

```
glRasterPos2i(rx, ry);
glutBitmapCharacter(GLUT_BITMAP_8_BY_13, k);
rx += glutBitmapWidth(GLUT_BITMAP_8_BY_13, k);
```

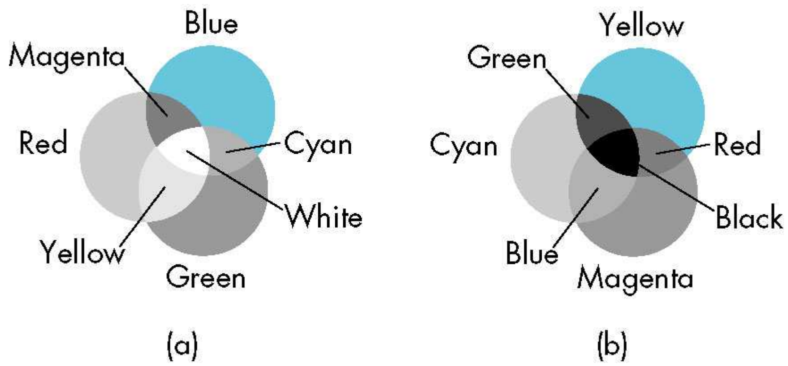
RGB: Additive Color Matching



The Color Solid (Color Cube)



Additive and Subtractive Color



RGB CMY (CMYK)

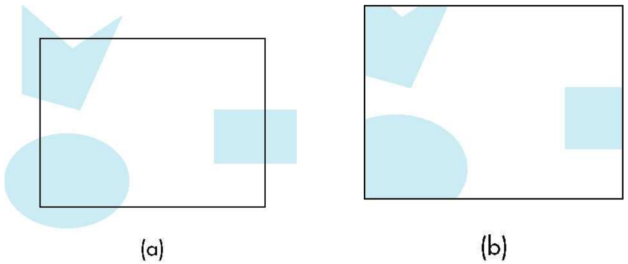
Outline for today

- OpenGL API and libraries 1×1
- Graphics primitives, attributes, colors
- Simple (orthographic) viewing
- Control and the window system
- 3D graphics

$\Rightarrow$ **Simple (orthographic) viewing**

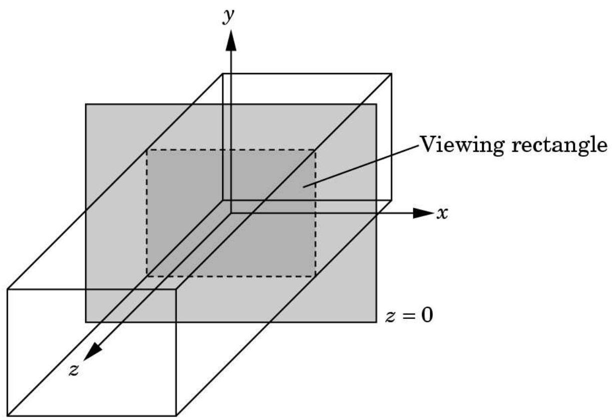
Viewing

Defining a relation between objects and camera
$\longrightarrow$ *perspective*
2D-viewing (just clipping): *viewing/clipping rectangle*

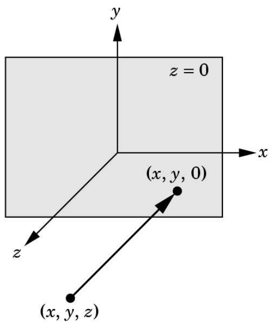


3D Viewing/Clipping

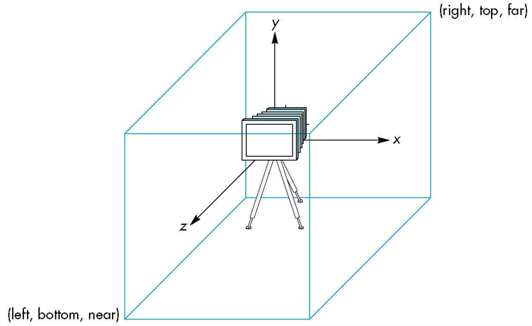
Viewing rectangle is $z = 0$
OpenGl default is $2 \times 2 \times 2$ volume
$(-1, -1, -1)$ to $(1, 1, 1)$



Orthographic View



Projection vectors are **orthogonal** to the projection plane. (From vertex (x, y, z) to $(x, y, 0)$.)



Default camera: also "sees" what is behind it.

Using `glOrtho`

```
void glOrtho(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top, GLdouble near, GLdouble far)

void gluOrtho2D(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top)
```

Matrix Modes

- OpenGL has two modes:
  - ⋆ changing projection
  - ⋆ drawing objects
- More in Lectures 4 and 5 (math)

```
glMatrixMode(GL_PROJECTION);
glLoadIdentity();
gluOrtho2D(0.0, 500.0, 0.0, 500.0);
glMatrixMode(GL_MODELVIEW);
```


Outline for today

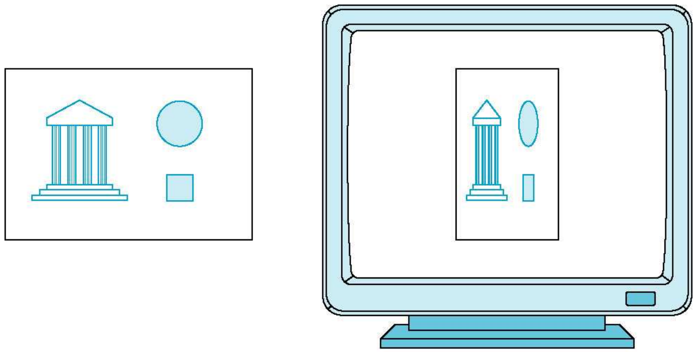
- OpenGL API and libraries 1×1
- Graphics primitives, attributes, colors
- Simple (orthographic) viewing
- Control and the window system
- 3D graphics

$\Rightarrow$ **Control and the window system**

Control and the Window System

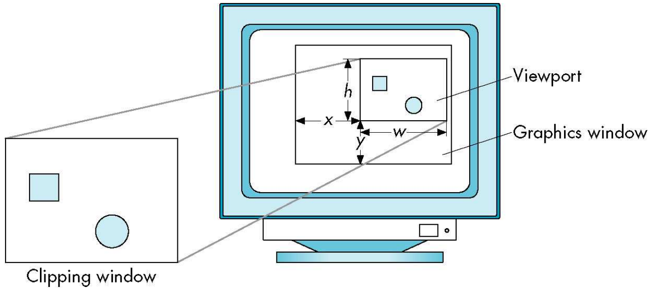
```
#include <GL/glut.h>
int main(int argc, char** argv){
  glutInit(&argc,argv);
  glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB);
  glutInitWindowSize(500,500);
  glutInitWindowPosition(0,0);
  glutCreateWindow("Sierpinski Gasket");
  glutDisplayFunc(display); /* register display func. */

  myinit();       /* application-specific inits */
  glutMainLoop(); /* enter event loop */
  return 0;
}
```


Window Size and Aspect Ratio



(a) (b)

Viewports




```
void glViewport( GLint x, GLint y, GLsizei w, GLsizei h)
```


myinit()

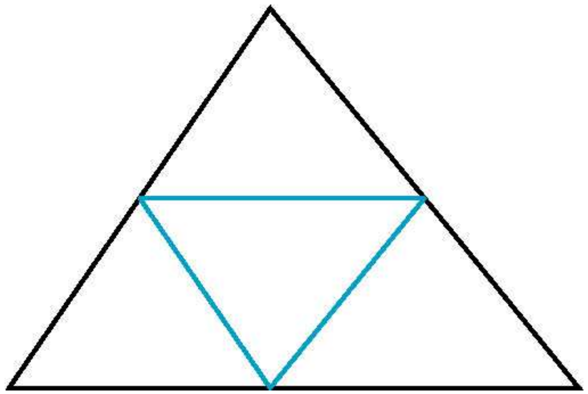
```
void myinit(void)
{
  glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */
                            /* ^----- opaque background */
  glColor3f(1.0, 0.0, 0.0); /* draw in red */

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  gluOrtho2D(0.0, 500.0, 0.0, 500.0);
  glMatrixMode(GL_MODELVIEW);
}
```

Outline for today

- OpenGL API and libraries 1×1
- Graphics primitives, attributes, colors
- Simple (orthographic) viewing
- Control and the window system
- 3D graphics

$\Rightarrow$ **3D graphics**

Sierpinski Gasket by Triangle Bisection



Drawing a Triangle

```
void triangle( point2 a, point2 b, point2 c){
  glBegin(GL_TRIANGLES);
     glVertex2fv(a);
     glVertex2fv(b);
     glVertex2fv(c);
  glEnd();
}
```

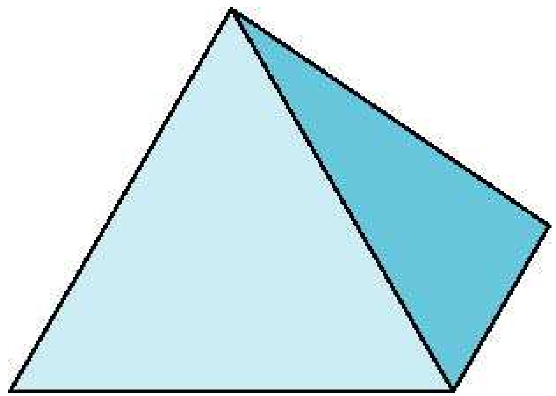

Dividing Triangles

```
void divide_triangle(point2 a, point2 b, point2 c, int m){
    point2 v0, v1, v2;
    int j;
    if(m>0) {
        for(j=0; j<2; j++) v0[j]=(a[j]+b[j])/2;
        for(j=0; j<2; j++) v1[j]=(a[j]+c[j])/2;
        for(j=0; j<2; j++) v2[j]=(b[j]+c[j])/2;
        divide_triangle(a, v0, v1, m-1);
        divide_triangle(c, v1, v2, m-1);
        divide_triangle(b, v2, v0, m-1);
    }
    else(triangle(a,b,c));
}
```

Display it

```
void display(void) {
    glClear(GL_COLOR_BUFFER_BIT);
    divide_triangle(v[0], v[1], v[2], n);
    glFlush();
}
```


The 3-Dimensional Gasket



myinit() for 3D gasket

```
void myinit(void){

  glClearColor(1.0, 1.0, 1.0, 1.0); /* white background */

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glOrtho(-500.0, 500.0, -500.0, 500.0, -500.0, 500.0);
  glMatrixMode(GL_MODELVIEW);
}
```


Drawing a 3D-Triangle

```
void triangle( point3 a, point3 b, point3 c){
  glBegin(GL_TRIANGLES);
    glVertex3fv(a);
    glVertex3fv(b);
    glVertex3fv(c);
 glEnd();
}
```

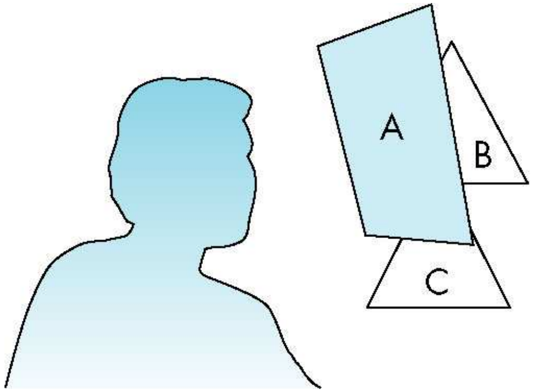
Dividing 3D-Triangles

```
void divide_triangle(point3 a, point3 b, point3 c, int m){
  point3 v0, v1, v2;
  int j;
  if(m>0){
    for(j=0; j<3; j++) v0[j]=(a[j]+b[j])/2;
    for(j=0; j<3; j++) v1[j]=(a[j]+c[j])/2;
    for(j=0; j<3; j++) v2[j]=(b[j]+c[j])/2;
    divide_triangle(a, v0, v1, m-1);
    divide_triangle(c, v1, v2, m-1);
    divide_triangle(b, v2, v0, m-1);
  }
  else(triangle(a,b,c));
}
```


`display()` for 3D with triangles

```
void display(void){  /* be n the recursion level */
  glClear(GL_COLOR_BUFFER_BIT);
  glColor3f(1.0,0.0,0.0);
  divide_triangle(v[0],v[2],v[3], n);
  glColor3f(0.0,1.0,0.0);
  divide_triangle(v[0],v[1],v[2], n);
  glColor3f(0.0,0.0,1.0);
  divide_triangle(v[1],v[2],v[3], n);
  glColor3f(0.0,0.0,0.0);
  divide_triangle(v[0],v[1],v[3], n);
  glFlush();
}
```

Hidden Surface Removal



Let's add Hidden-Surface Removal

```
int main(int argc, char **argv) {
  if ( argc < 2 ) { printf("synopsis: %s <recursion depth>\n",argv[0]); }
  else{
    n=atoi(argv[1]);
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_SINGLE | GLUT_RGB | GLUT_DEPTH );
    glutInitWindowSize(500, 500);
    glutCreateWindow("3D Gasket, Triangles, hidden-surface removal");
    glutDisplayFunc(display);
    glEnable(GL_DEPTH_TEST);
    myinit();
    glutMainLoop();
  }
  return 0;
}
```

. . . and don't forget:

```
void display(void){
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  ...
}
```

with

without

Summary

What to remember:

- Vertices make geometric objects
- Categories of graphics functions
- RGB color
- Orthographic viewing
- Using the GLUT library
- Hidden-surface removal

Next week:

- CAVE excursion

Next lecture:

- Input and interaction