

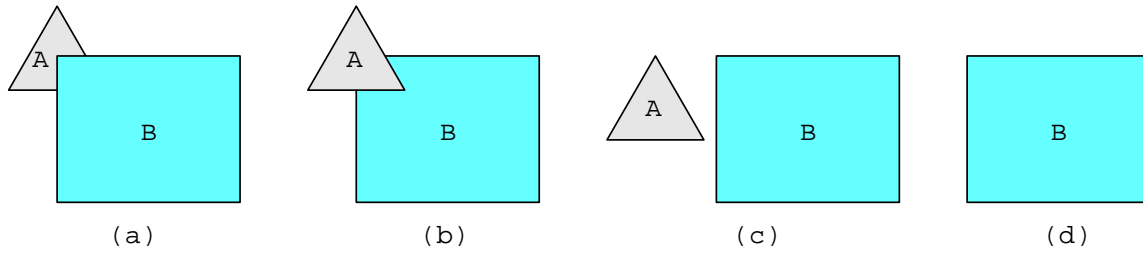
Why polygon restrictions?

- non-convex and non-simple polygons are expensive to process and to render
- convexity and simplicity are expensive to test
- better to fix polygons as a pre-processing step
- some tools in GLU to do this (e.g., tessellations)
- behavior of OpenGL implementation on disallowed polygons is “undefined”
- triangles are most efficient in hardware

Hidden-surface removal:

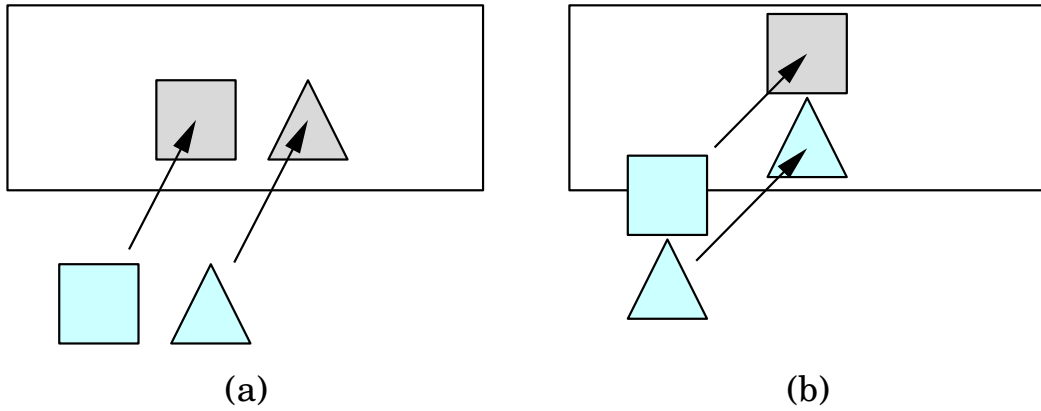
- what is visible after clipping and projection?
- object-space vs. image-space approaches
- object space: depth sort (Painter's alg)
- image space: ray cast (z buffer alg)
- much more in Ch8

Object-space approach:



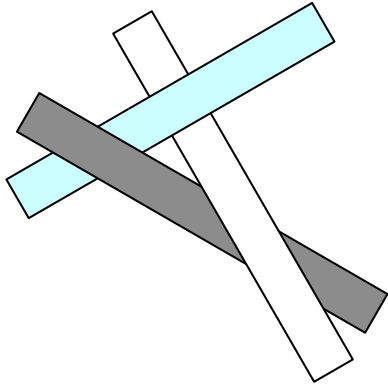
- consider pairs of objects; do they occlude?
- complexity $O(k^2)$, where $k =$ number of objects
- Painter's algorithm: render back-to-front
- ...but how to *do* that sort?

Depth sorting:

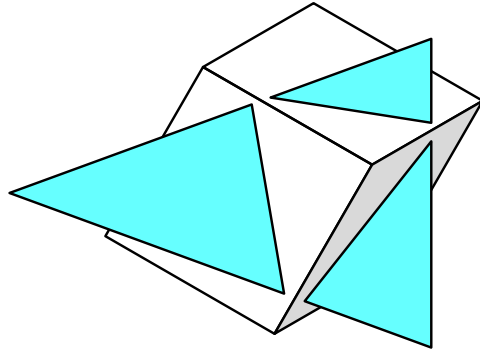


- first sort by furthest distance z from the viewer
- if minimum depth of A is greater than maximum depth of B, A can be drawn before B
- if either x or y extents do not overlap, A and B can be drawn independently

Difficult cases:



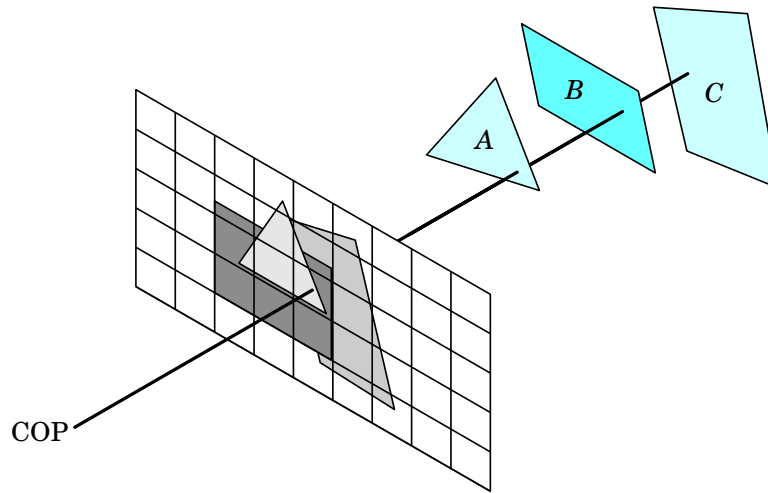
cyclic overlap



piercing polygons

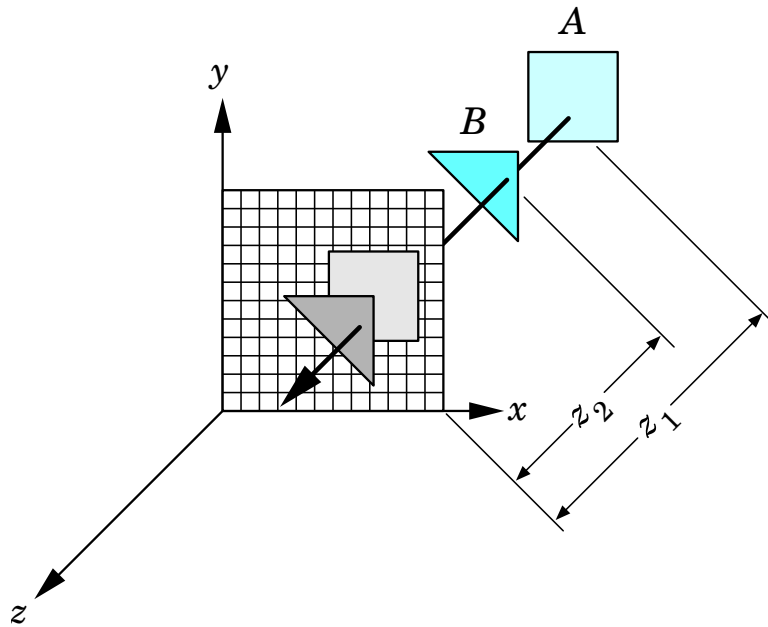
- → sometimes cannot sort polygons
- one solution: compute intersections and subdivide (ouch)

Image-space approach:



- raycasting: intersect ray with polygons
- $O(k)$ worst case (though often better)

The z-buffer algorithm:



- z-buffer with depth value z for each pixel
- before writing a pixel into frame buffer
 - compute distance z of pixel origin from viewer
 - if closer write and update z-buffer; otherwise discard

HSR in OpenGL with the Z buffer:

Request auxiliary storage:

```
glutInitDisplayMode (GLUT_SINGLE | GLUT_RGB  
| GLUT_DEPTH);
```

...and enable the algorithm:

```
glEnable(GL_DEPTH_TEST);
```

(both in main)

NB: have to clear **it** too, so `glClear` becomes:

```
glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
```

Types of callbacks:

- `display()`: when window must be drawn
- `idle()`: when no other events to be handled
- `keyboard(unsigned char key, int x, int y)`: when key is struck
- `menu(...)`: after selection from menu
- `mouse(int button, int state, int x, int y)`: when mouse is clicked
- `motion(...)`: when mouse is moved
- `reshape(int w, int h)`: when window is resized
- ...any callback can be NULL

Reshape callback:

```
void reshape(GLsizei w, GLsizei h)
{
    /* adjust clipping box */
    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    gluOrtho2D(0.0, 500.0, 0.0, 500.0);
    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();

    /* adjust viewport and clear */
    glViewport(0,0,w,h);
    glClearColor(0.0,0.0,0.0,0.0);
    glClear(GL_COLOR_BUFFER_BIT);
    glFlush();
}
```

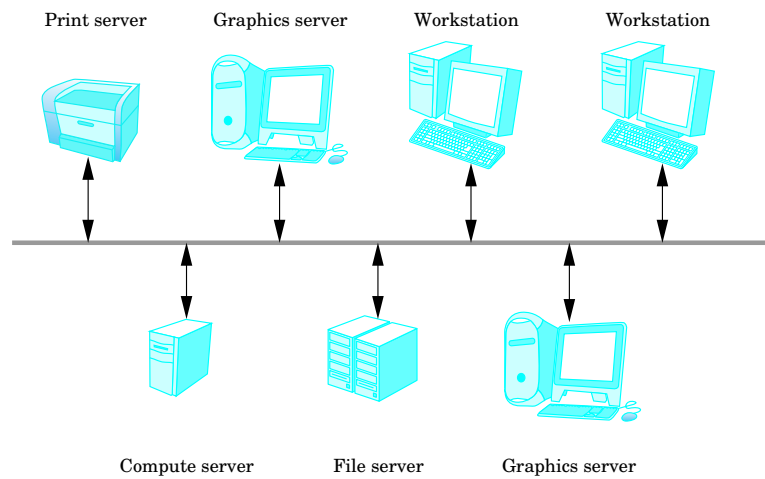
(adjusts all coord systems after a reshape so as to keep given polygons same size and shape)

More on the display callback:

- required, though can be NULL (be careful! *)
- don't call directly if you need to redisplay; rather, use `glutPostRedisplay()`; to set OpenGL's internal "redraw the window" flag
- can display to multiple windows:
 - open with `id=glutCreateWindow('another window')`;
 - (use `glutInitDisplayMode` beforehand if want different properties)
 - select with `glutSetWindow(id)`;
 - *NB: state!*

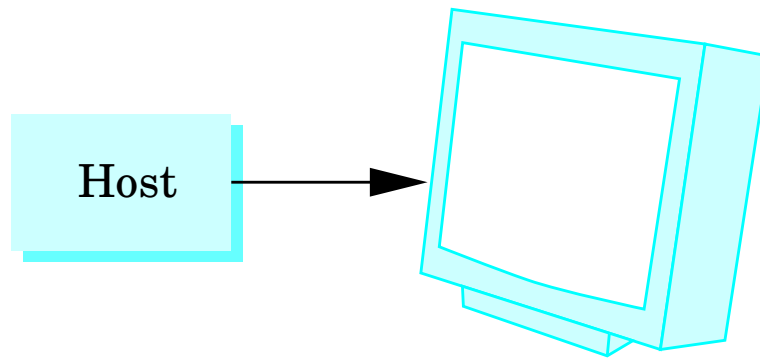
* cf., `square.c`

Graphics habitat:

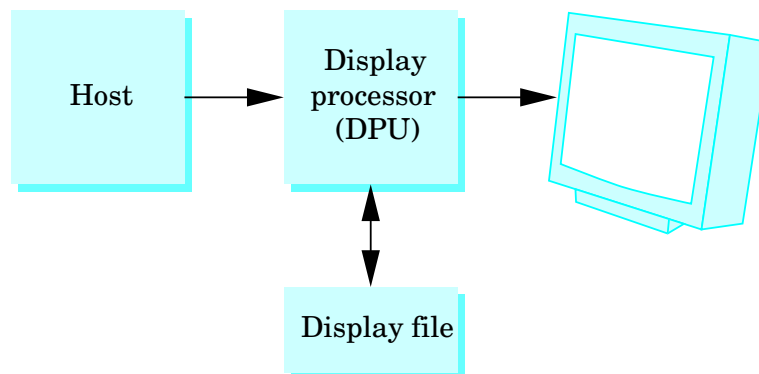


- extremely intensive task
- special hardware, caching, parallelism, ...
- but means that you have to know where data live
- and events can come from all directions at any time.

Graphics architectures:



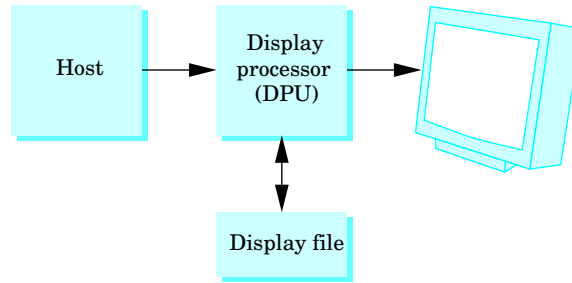
then



now

... "client-server"

The display processor:



- DPU has limited instruction set
- user pgm compiled on host, producing **display list**
- display list sent to DPU, freeing host to do other stuff
- DPU loops on that display list

DPU in “immediate-mode” graphics, while host sees “retained-mode” graphics.

Display lists:

Encapsulate a sequence of drawing commands:

```
// unique integer identifier:
#define BOX 1

// a red box:
glNewList(BOX, GL_COMPILE);
    glBegin(GL_POLYGON);
        glColor3f(1.0, 0.0, 0.0);
        glVertex2f(-1.0, -1.0);
        glVertex2f(1.0, -1.0);
        glVertex2f(1.0, 1.0);
        glVertex2f(-1.0, 1.0);
    glEnd();
glEndList();
```

GL_COMPILE: just send list to server

GL_COMPILE_AND_EXECUTE: send *and* display

To use: `glCallList(BOX);`

Working with multiple display lists: `glGenLists`,
`glCallLists`; see p102.

More about display lists:

- useful for sequences of transformations
- important for complex surfaces
- hierarchical display lists supported
- display lists cannot be changed...
- ...but they can be replaced.

Display list caveats:

Remember that OpenGL is stateful...

```
glMatrixMode(GL_PROJECTION);  
for (i=1; i<5; i++)  
{  
    glLoadIdentity();  
    gluOrtho2D(-2.0*i, 2.0*i, -2.0*i, 2.0*i);  
    glCallList(BOX);  
}
```

What does this do?

May not be a good idea to change state vars inside a display list.

But that would reduce their effectiveness...

Stacks.

- matrix (e.g., GL_PROJECTION)
- attribute (e.g., color)

Push old one; frob current one as desired;
draw; pop.

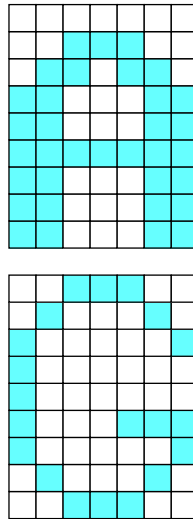
To use:

```
glPushAttrib(GL_ALL_ATTRIB_BITS);  
glPushMatrix();  
...frob...  
glPopMatrix();  
glPopMatrix();
```

NB: similar to `glClear` *et al.* — specify target, then op.

Text:

raster:



stroke:

Computer:
Graphics

Raster text:

- bit blocks
- simple and fast (e.g., `bitblt` transfer)
- awkward to magnify:



Raster fonts in OpenGL:

OpenGL provides a *few* bitmapped character sets:

```
glutBitmapCharacter(GLUT_BITMAP_8_BY_13, int  
character);
```

Placed at “current raster position,” which you can change with `glRasterPos*`.

Current raster position (CRP) moves one char right after `glutBitmapCharacter`.

Can keep explicit track of CRP if you want:

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

Can query fontsize with `glutBitmapWidth`.

Stroke fonts in OpenGL:

OpenGL also provides a few *stroke* character sets:

```
glutStrokeCharacter(GLUT_STROKE_MONO_ROMAN,  
int character);
```

These are polygons, and are subject to all the transformations, etc., in the pipeline.

Arbitrary wierd size; use `glPushMatrix` and `glPopMatrix` to scale.

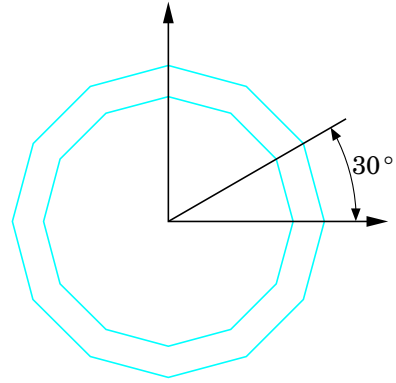
Positioning, needless to say, is a bear; CRP doesn't help...

(Can also use fonts provided by windowing system — but at the expense of portability.)

Display lists and stroke fonts:

```
void MyFont(char c)
{
    switch(c)
    {
        case 'a':
            ...
            break;
        case 'A':
            ...
            break;
        ...
    }
}
```

Display lists and stroke fonts, cont.:



```
case '0':
    /* move to center */
    glTranslatef(0.5,0.5,0.0);

    glBegin(GL_QUAD_STRIP);

    /* 12 vertices */
    for (i=0; i<=12; i++)
    {
        angle = M_PI / 6.0 * i;
        glVertex2f(0.4*cos(angle),
                  0.4*sin(angle));
        glVertex2f(0.5*cos(angle),
                  0.5*sin(angle));
    }
    glEnd();

    /* move to lower right */
    glTranslatef(0.5,-0.5,0.0);

    break;
```

Display lists and stroke fonts, cont.:

```
/* return index of first of 256
   consecutive available ids */
base = glGenLists(256);

for (i=0; i<256; i++)
{
    glNewList(base+i, GL_COMPILE);
    myFont(i);
    glEndList();
}
```