3D graphics with Perl

Jonathan Chin <jon-techtalk@earth.li>

November 2005

OpenGL

- API for real-time 3D graphics

OpenGL

- API for real-time 3D graphics
- Abstracts away the details of complicated rendering hardware/software

OpenGL

- API for real-time 3D graphics
- Abstracts away the details of complicated rendering hardware/software
- Standard on UNIX; competes with Direct3D on Windows.

- OpenGL will:
  - Render 3d graphics very quickly.

- OpenGL will:
  - Render 3d graphics very quickly.
  - Get the hardware to do it, if possible.

- OpenGL will:
  - Render 3d graphics very quickly.
  - Get the hardware to do it, if possible.
- OpenGL will not:

- OpenGL will:
  - Render 3d graphics very quickly.
  - Get the hardware to do it, if possible.
- OpenGL will not:
  - Raytrace

- OpenGL will:
  - Render 3d graphics very quickly.
  - Get the hardware to do it, if possible.
- OpenGL will not:
  - Raytrace
  - Deal with windowing, input, mouse/keyboard interaction

Some graphics hardware from 2002

Equivalent hardware in 2005

Talking to OpenGL from Perl

- `OpenGL.pm`

Talking to OpenGL from Perl

- `OpenGL.pm`
- `SDL::OpenGL`

Talking to OpenGL from Perl

- `OpenGL.pm`
- `SDL::OpenGL`
- `OpenGL::Simple`

Step 1: Open a window.

- X11 (The `OpenGL.pm` way)

Step 1: Open a window.

- X11 (The `OpenGL.pm` way)
- SDL

Step 1: Open a window.

- X11 (The `OpenGL.pm` way)
- SDL
- GTK

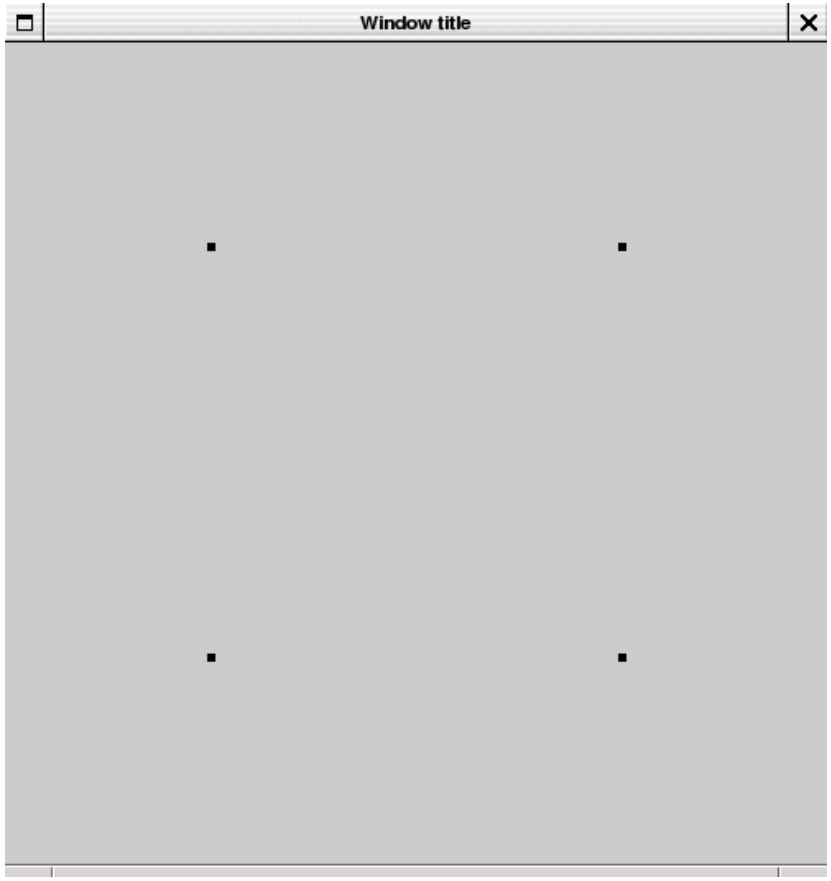
Step 1: Open a window.

- X11 (The `OpenGL.pm` way)
- SDL
- GTK
- GLUT

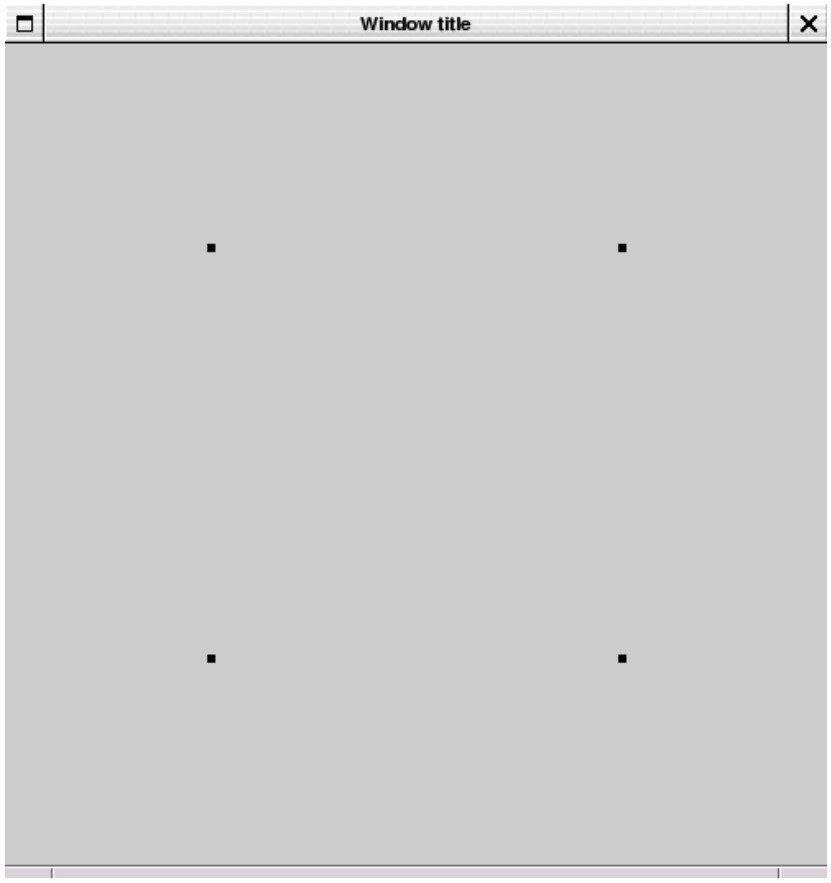
Step 1: Open a window.

- X11 (The `OpenGL.pm` way)
- SDL
- GTK
- GLUT
- `OpenGL::Simple::Viewer`

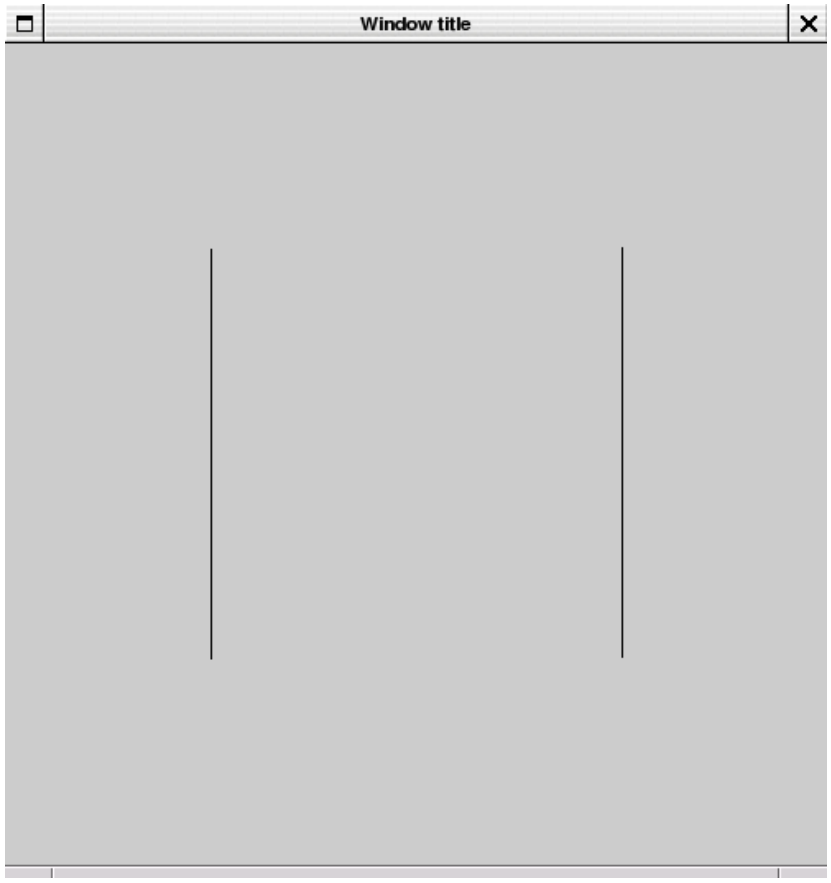
```
use OpenGL::Simple ':all';
use OpenGL::Simple::GLUT ':all';

glutInit;
glutInitWindowSize(500,500);
glutCreateWindow("Window title");
glutDisplayFunc(\&displayfunc);
glutMainLoop;

sub displayfunc {
    glClearColor(0.8, 0.8, 0.8, 0);
    glClear(GL_COLOR_BUFFER_BIT);
    glPointSize(5.0);
    glColor(0,0,0);
    glBegin(GL_POINTS);
        glVertex(-0.5, -0.5);
        glVertex(-0.5,  0.5);
        glVertex( 0.5,  0.5);
        glVertex( 0.5, -0.5);
    glEnd;
    glFlush;
}
```



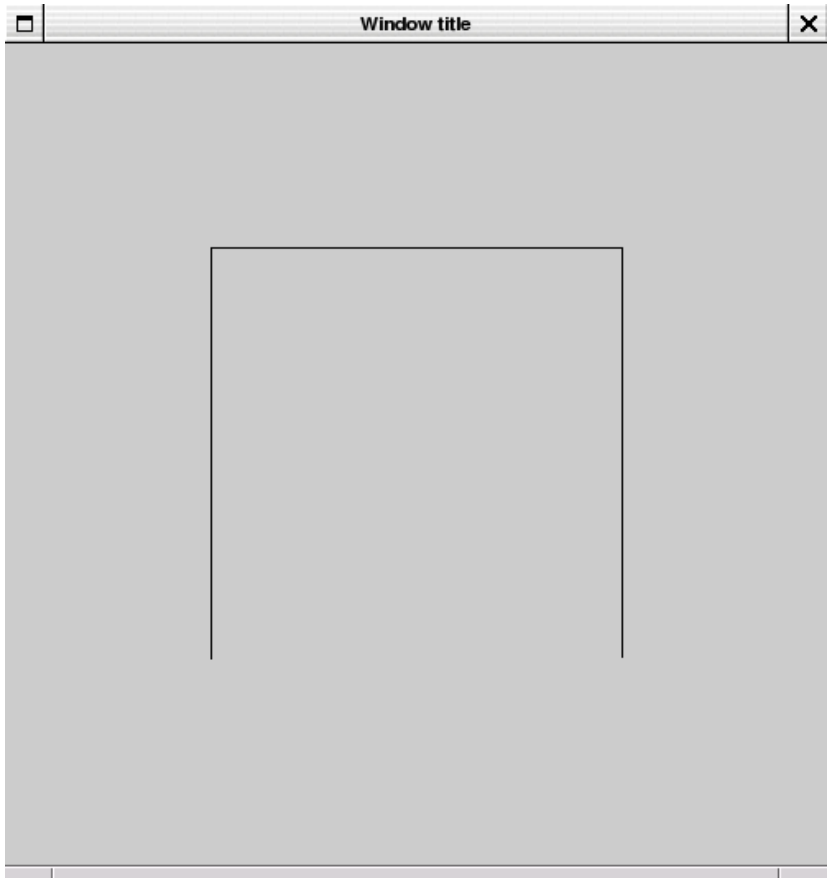
```
glBegin(GL_POINTS);
    glVertex(-0.5, -0.5);
    glVertex(-0.5,  0.5);
    glVertex( 0.5,  0.5);
    glVertex( 0.5, -0.5);
glEnd;
```



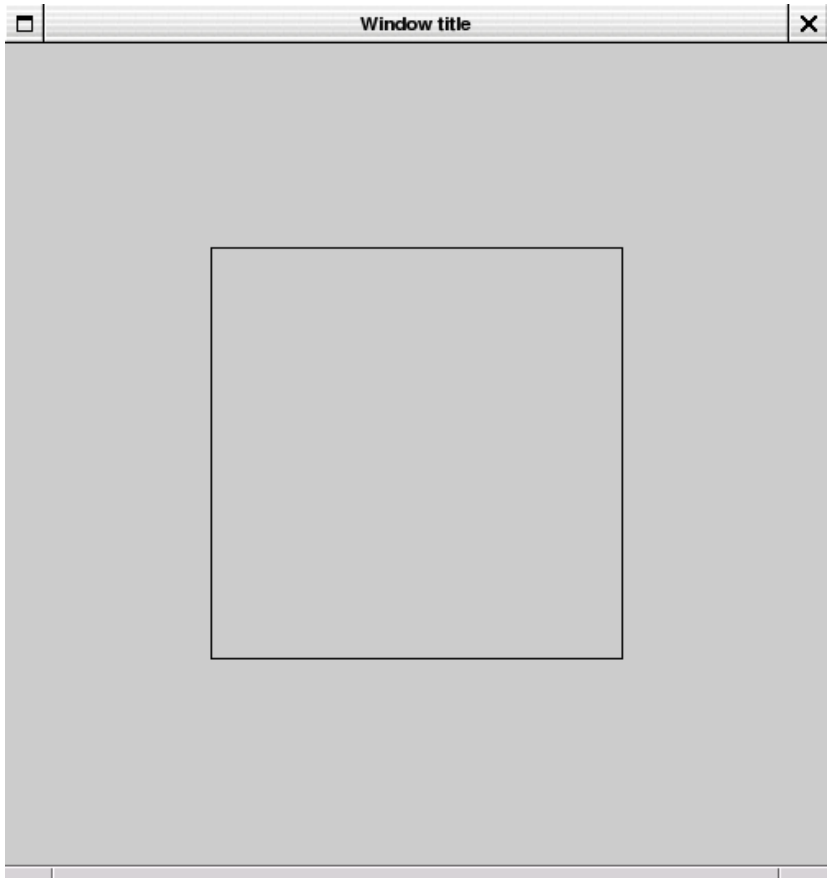
```
glBegin(GL_LINES);
    glVertex(-0.5, -0.5);
    glVertex(-0.5,  0.5);
    glVertex( 0.5,  0.5);
    glVertex( 0.5, -0.5);
glEnd;
```



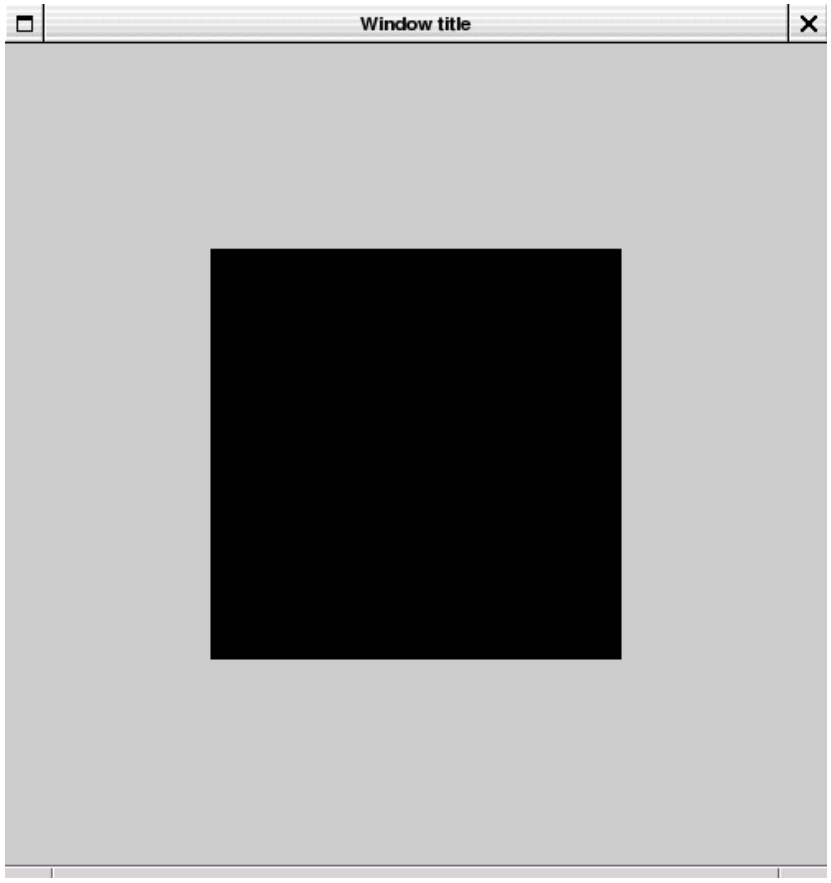
```
glBegin(GL_LINE_STRIP);
    glVertex(-0.5, -0.5);
    glVertex(-0.5,  0.5);
    glVertex( 0.5,  0.5);
    glVertex( 0.5, -0.5);
glEnd;
```



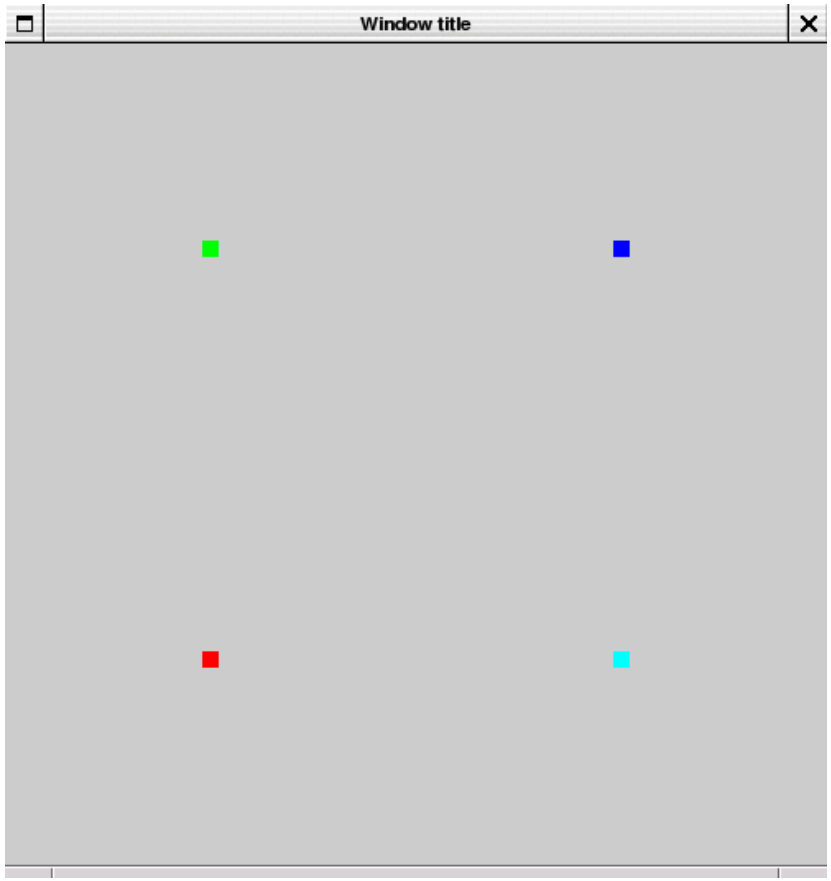
```
glBegin(GL_LINE_LOOP);
    glVertex(-0.5, -0.5);
    glVertex(-0.5,  0.5);
    glVertex( 0.5,  0.5);
    glVertex( 0.5, -0.5);
glEnd;
```



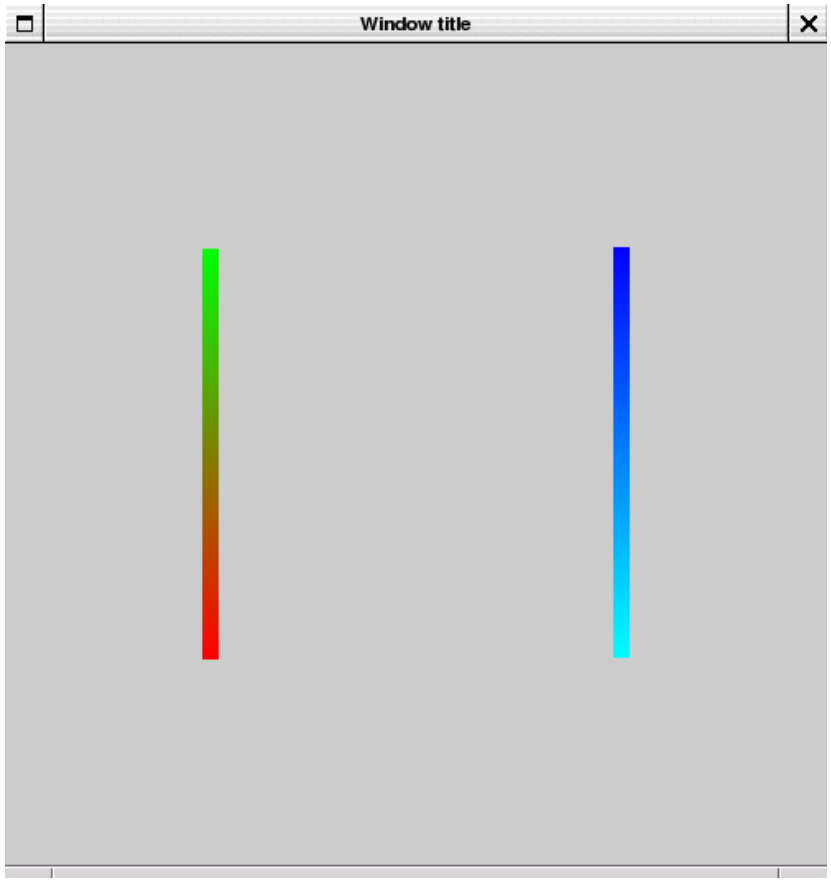
```
glBegin(GL_POLYGON);
    glVertex(-0.5, -0.5);
    glVertex(-0.5,  0.5);
    glVertex( 0.5,  0.5);
    glVertex( 0.5, -0.5);
glEnd;
```



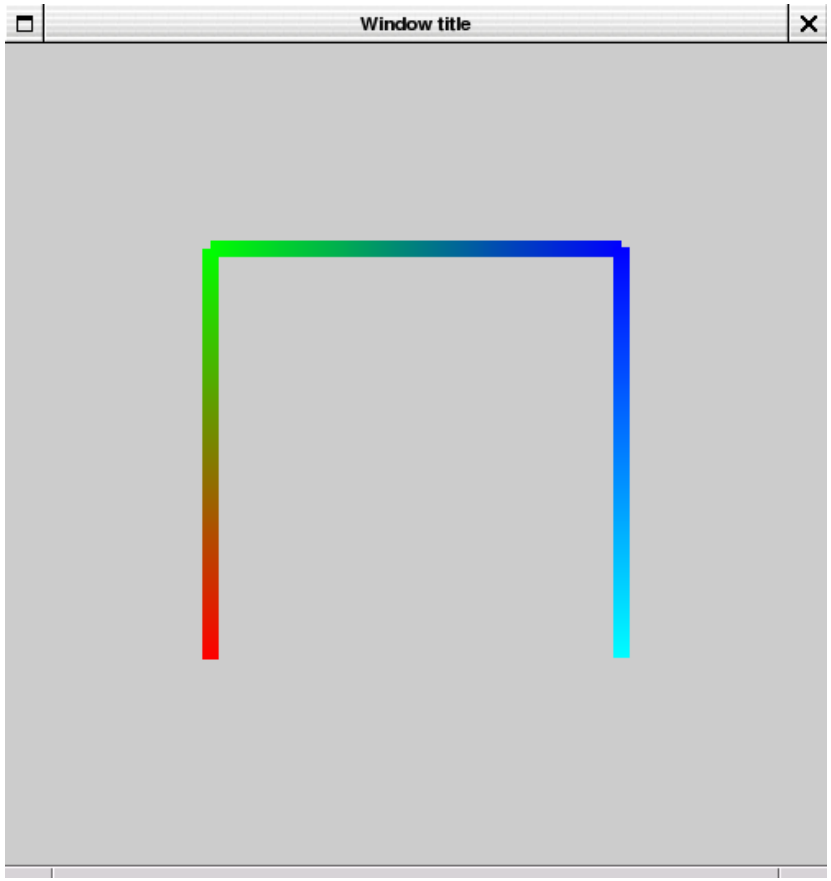
```
glBegin(GL_POINTS);
    glColor(1,0,0); glVertex(-0.5,-0.5);
    glColor(0,1,0); glVertex(-0.5,0.5);
    glColor(0,0,1); glVertex(0.5,0.5);
    glColor(0,1,1); glVertex(0.5,-0.5);
glEnd;
```



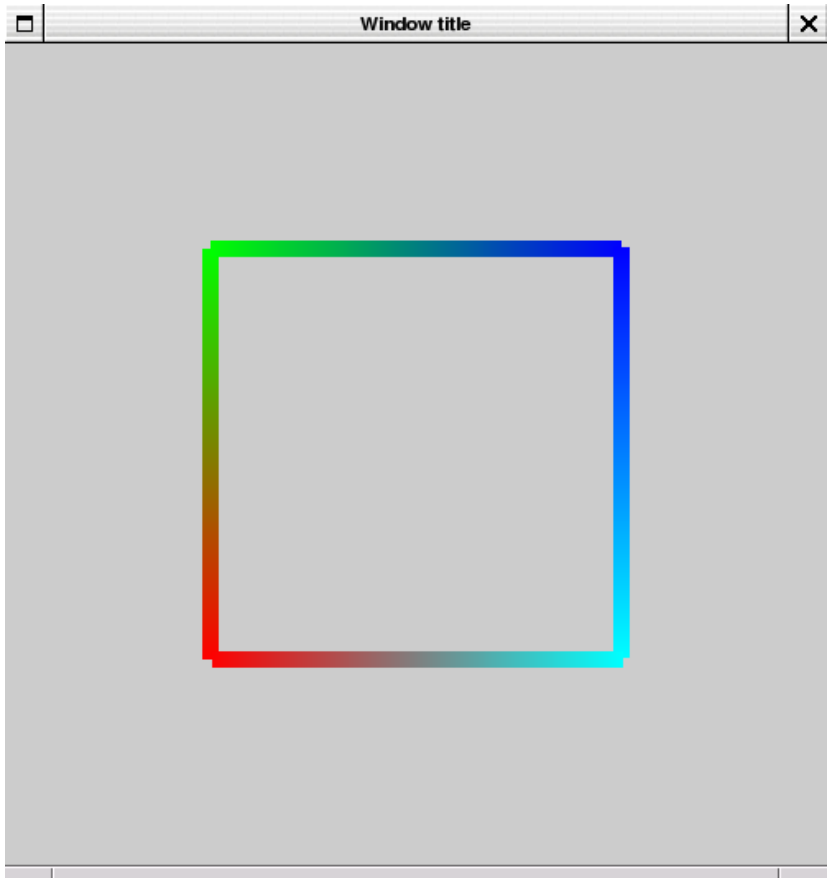
```
glBegin(GL_LINES);
    glColor(1,0,0); glVertex(-0.5,-0.5);
    glColor(0,1,0); glVertex(-0.5,0.5);
    glColor(0,0,1); glVertex(0.5,0.5);
    glColor(0,1,1); glVertex(0.5,-0.5);
glEnd;
```



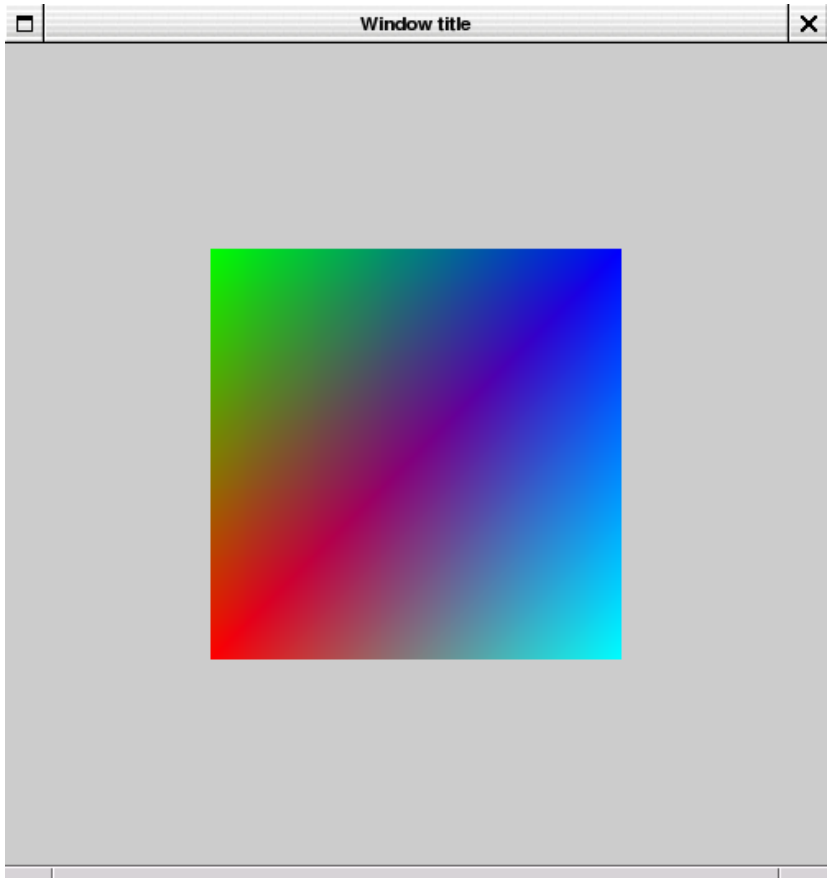
```
glBegin(GL_LINE_STRIP);
    glColor(1,0,0); glVertex(-0.5,-0.5);
    glColor(0,1,0); glVertex(-0.5,0.5);
    glColor(0,0,1); glVertex(0.5,0.5);
    glColor(0,1,1); glVertex(0.5,-0.5);
glEnd;
```



```
glBegin(GL_LINE_LOOP);
    glColor(1,0,0); glVertex(-0.5,-0.5);
    glColor(0,1,0); glVertex(-0.5,0.5);
    glColor(0,0,1); glVertex(0.5,0.5);
    glColor(0,1,1); glVertex(0.5,-0.5);
glEnd;
```

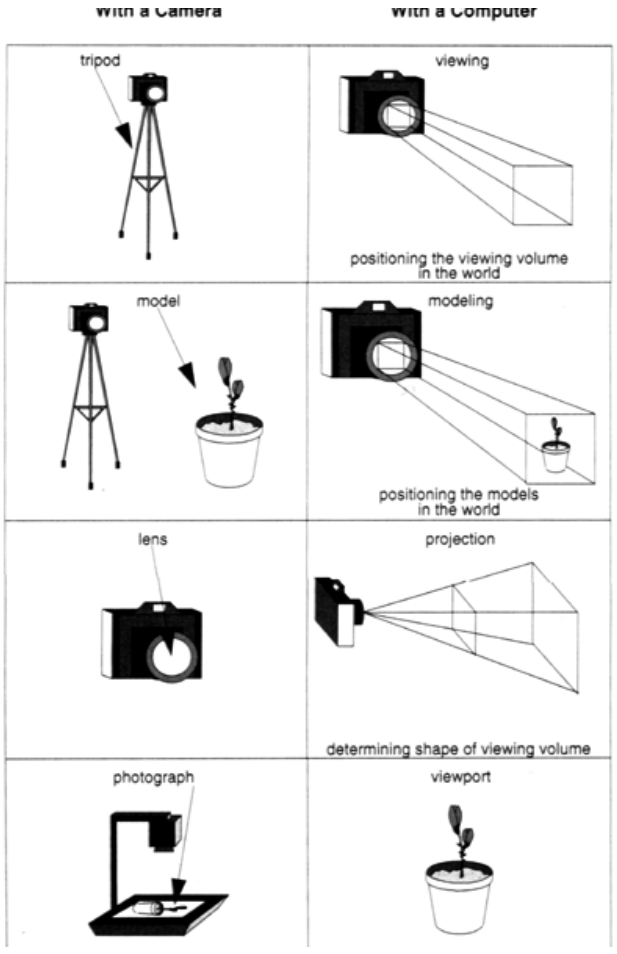


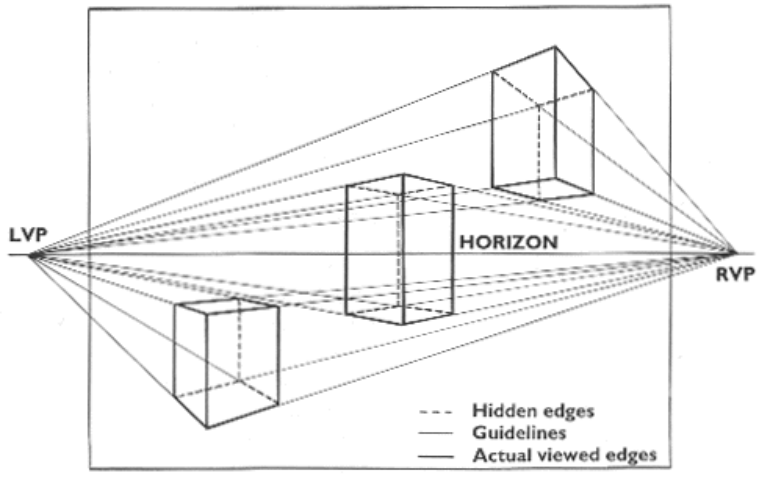
```
glBegin(GL_POLYGON);
    glColor(1,0,0); glVertex(-0.5,-0.5);
    glColor(0,1,0); glVertex(-0.5,0.5);
    glColor(0,0,1); glVertex(0.5,0.5);
    glColor(0,1,1); glVertex(0.5,-0.5);
glEnd;
```

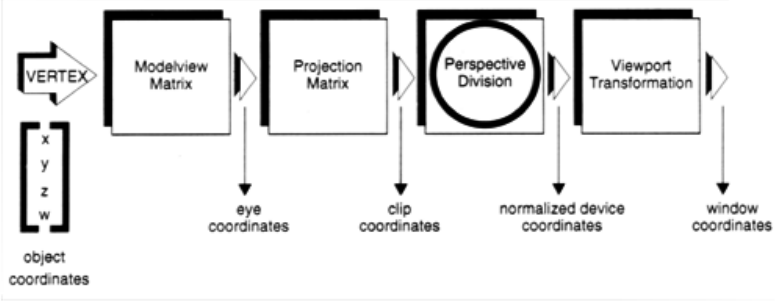


But what about **3D** graphics?

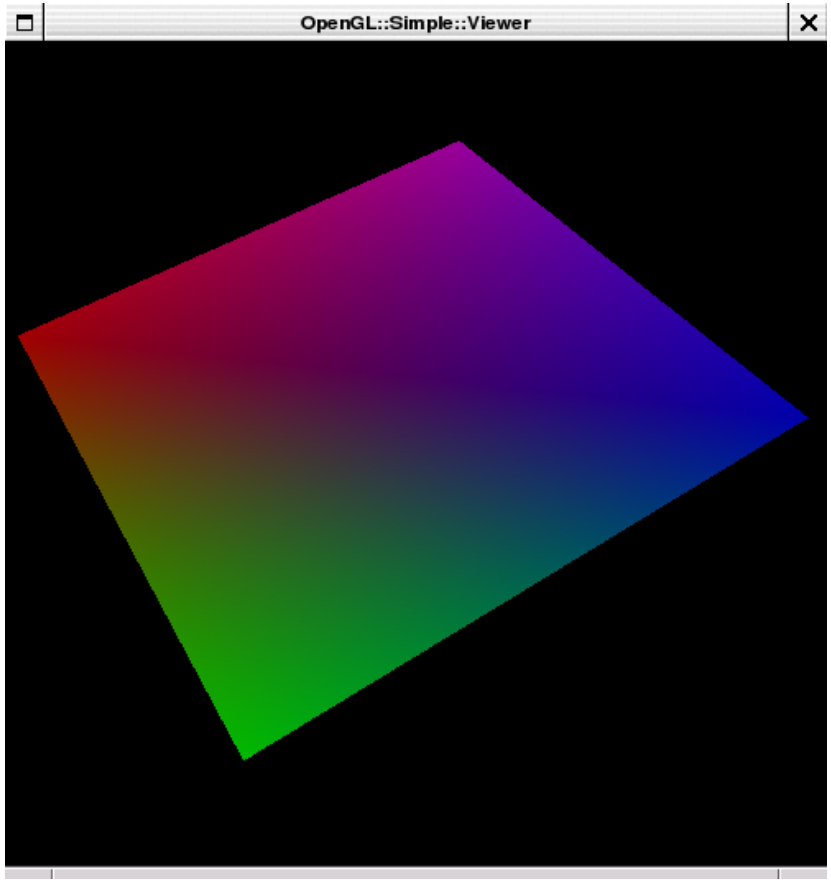
But what about **3D** graphics?



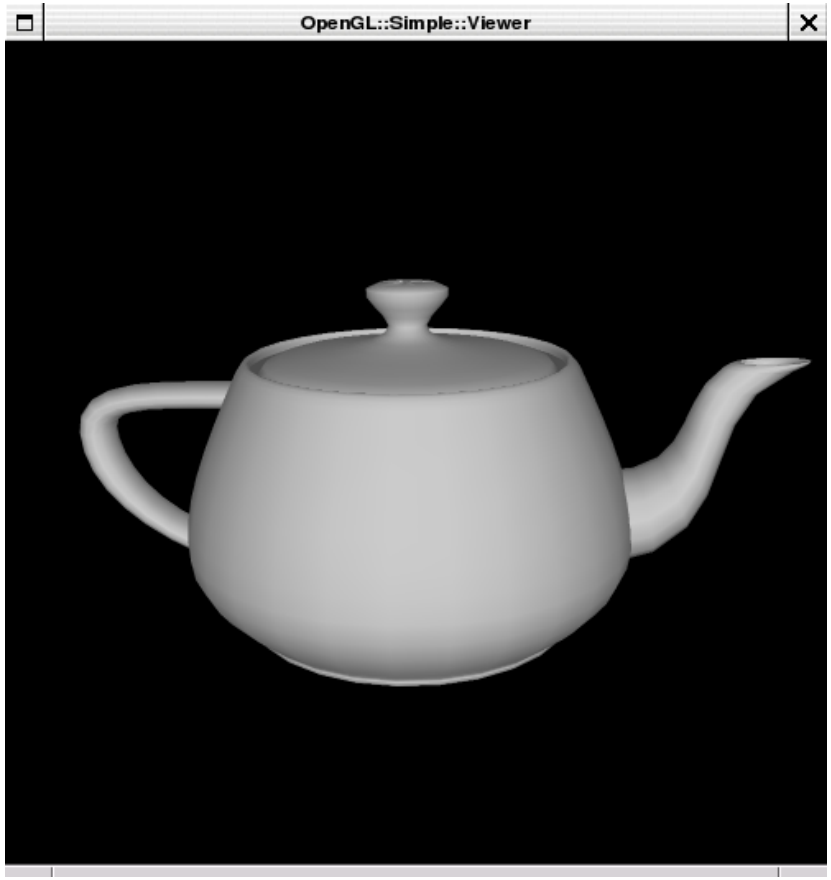




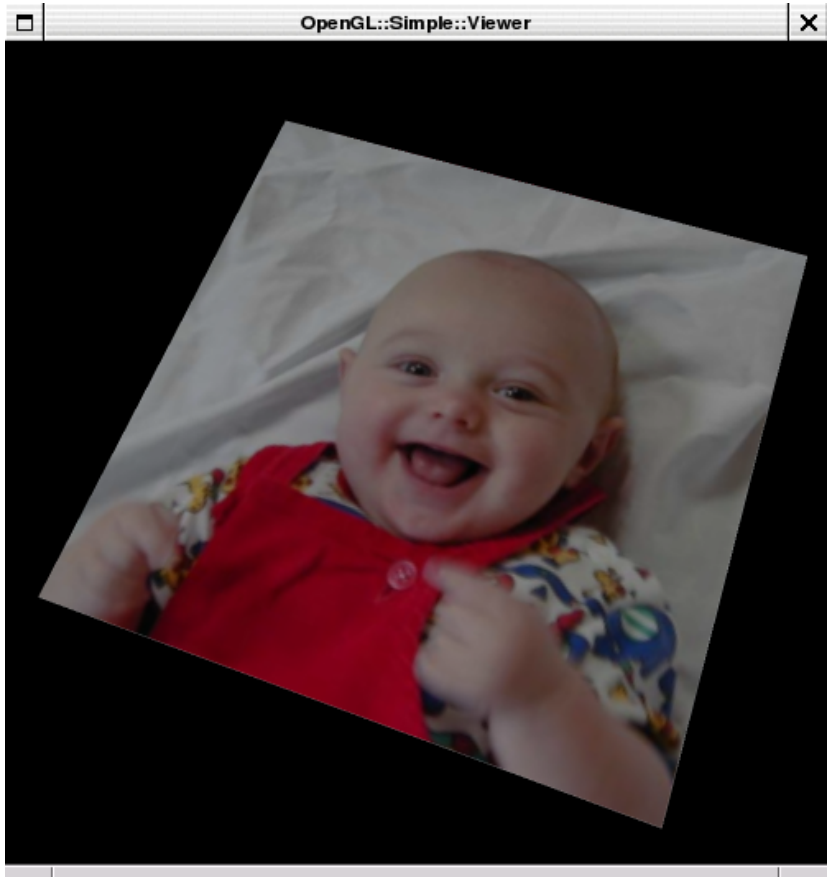
```
use OpenGL::Simple ':all';
use OpenGL::Simple::GLUT ':all';
use OpenGL::Simple::Viewer;

glutInit;

my $v = new OpenGL::Simple::Viewer(
  draw_geometry => sub {
    glBegin(GL_POLYGON);
      glColor(1,0,0); glVertex(-1,-1,0);
      glColor(0,1,0); glVertex( 1,-1,0);
      glColor(0,0,1); glVertex( 1, 1,0);
      glColor(1,0,1); glVertex(-1, 1,0);
    glEnd;
  },
);
glutMainLoop;
```



```
use OpenGL::Simple ':all';
use OpenGL::Simple::GLUT ':all';
use OpenGL::Simple::Viewer;

glutInit;

my $v = new OpenGL::Simple::Viewer(
    draw_geometry => sub {
        glutSolidTeapot(1.0);
    },
);

glutMainLoop;

exit;
```

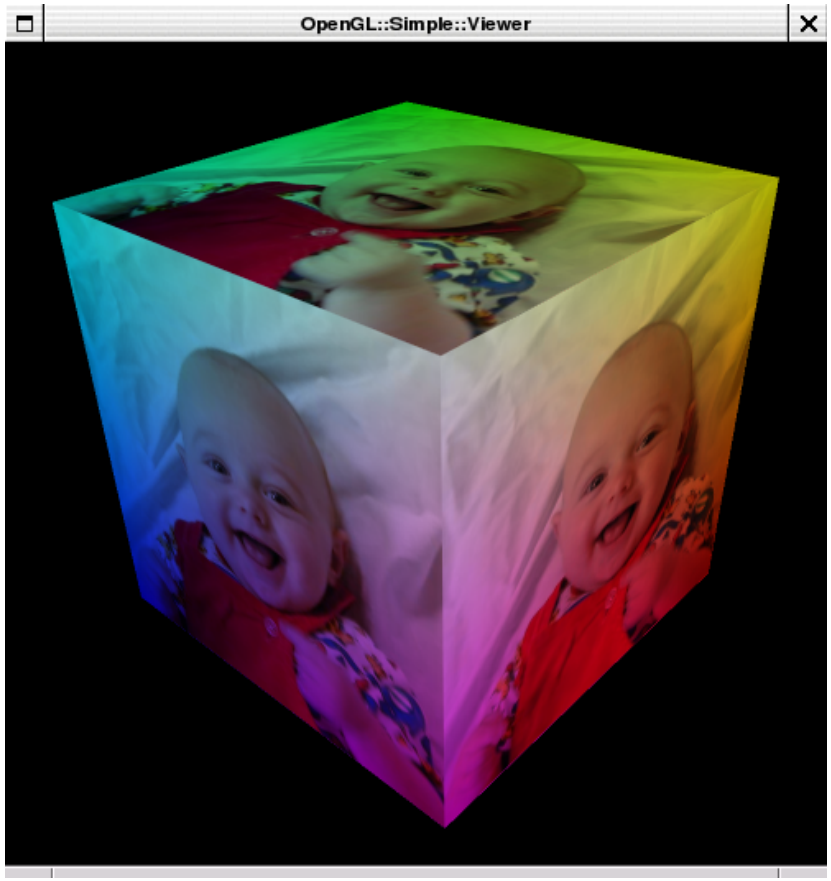


```
glutInit;

my $v = new OpenGL::Simple::Viewer(
  draw_geometry => sub {
    glBegin(GL_POLYGON);
       glTexCoord(0,1); glVertex(-1,-1,0);
       glTexCoord(1,1); glVertex( 1,-1,0);
       glTexCoord(1,0); glVertex( 1, 1,0);
       glTexCoord(0,0); glVertex(-1, 1,0);
    glEnd;
  },
);

my $i = new Imager;
$i->read(file=>'texture.png');
glTexImage2D(image=>$i);

glEnable(GL_TEXTURE_2D);
glTexParameter(GL_TEXTURE_2D,
    GL_TEXTURE_MAG_FILTER,GL_LINEAR);
glTexParameter(GL_TEXTURE_2D,
    GL_TEXTURE_MIN_FILTER,GL_LINEAR);

glutMainLoop;
```



```
glBegin(GL_QUADS);
 glTexCoord(0.0,1.0);
 glVertex(-1.0,-1.0, 1.0);

 glTexCoord(1.0,1.0);
 glVertex( 1.0,-1.0, 1.0);

 ...
glEnd;
```

```
glBegin(GL_QUADS);
 glColor(  0.0, -1.0,  1.0);
 glTexCoord(0.0, 1.0);
 glVertex(-1.0, -1.0,  1.0);

 glColor(  1.0,  0.0,  1.0);
 glTexCoord(1.0, 1.0);
 glVertex( 1.0, -1.0,  1.0);

...
glEnd;
```

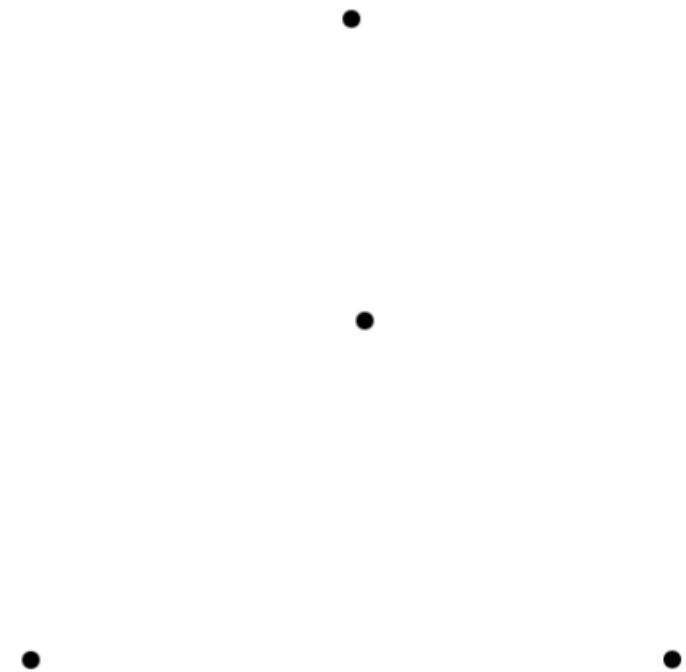


How to draw a simple fractal

Define three fixed points.

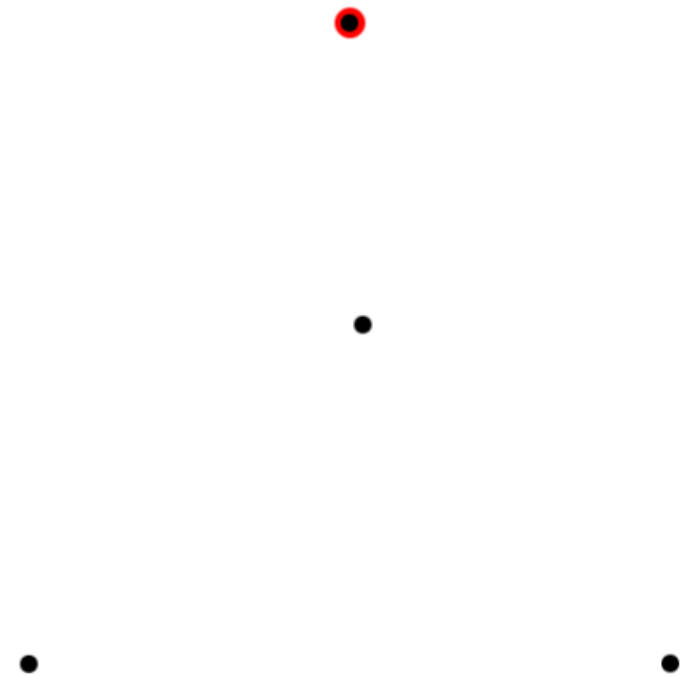
How to draw a simple fractal

Put a dot somewhere in the centre.



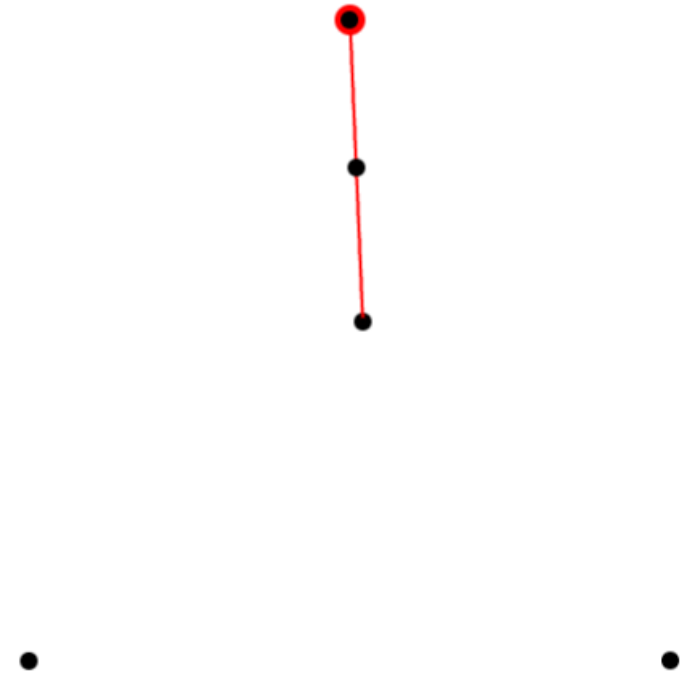
How to draw a simple fractal

Choose one of the fixed points, at random.



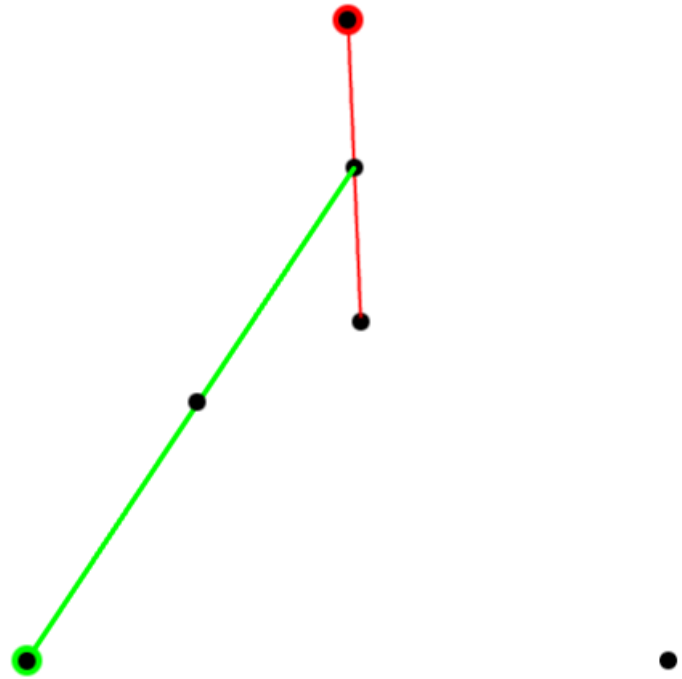
How to draw a simple fractal

Draw another dot, half-way between the previous dot and the fixed point.



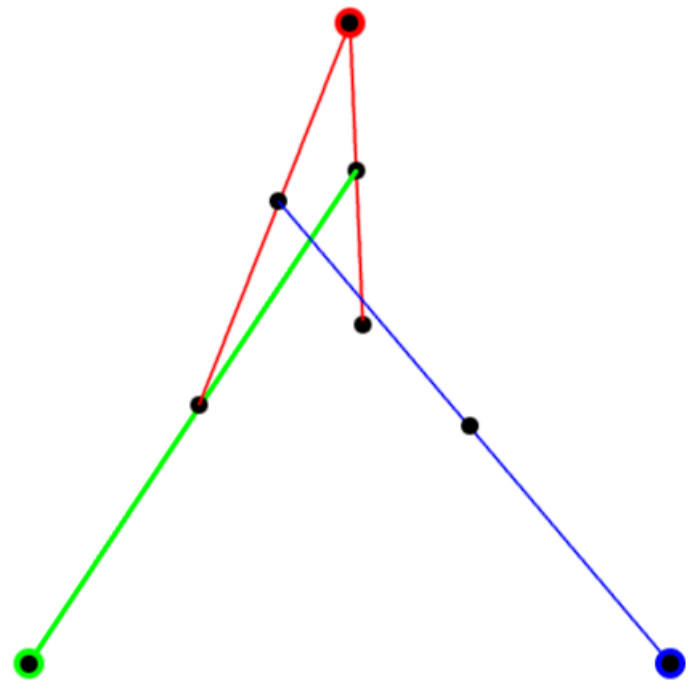
How to draw a simple fractal

Repeat.



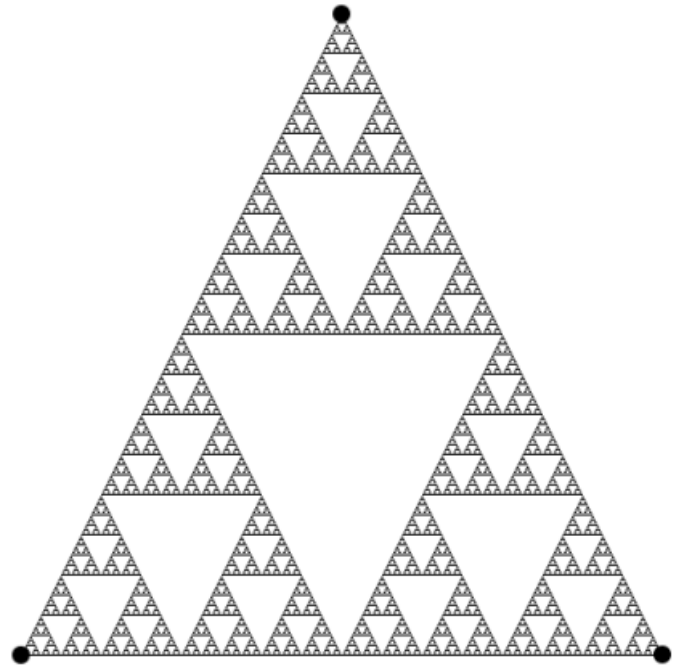
How to draw a simple fractal

Repeat many times.



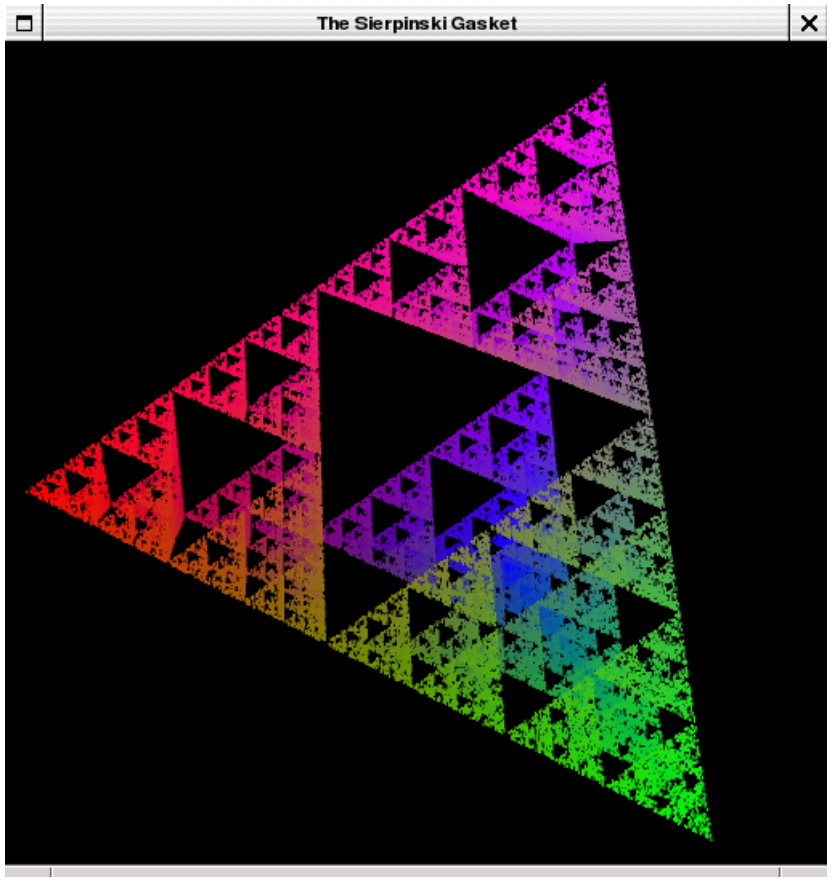
How to draw a simple fractal

Repeat many many times with smaller dots.

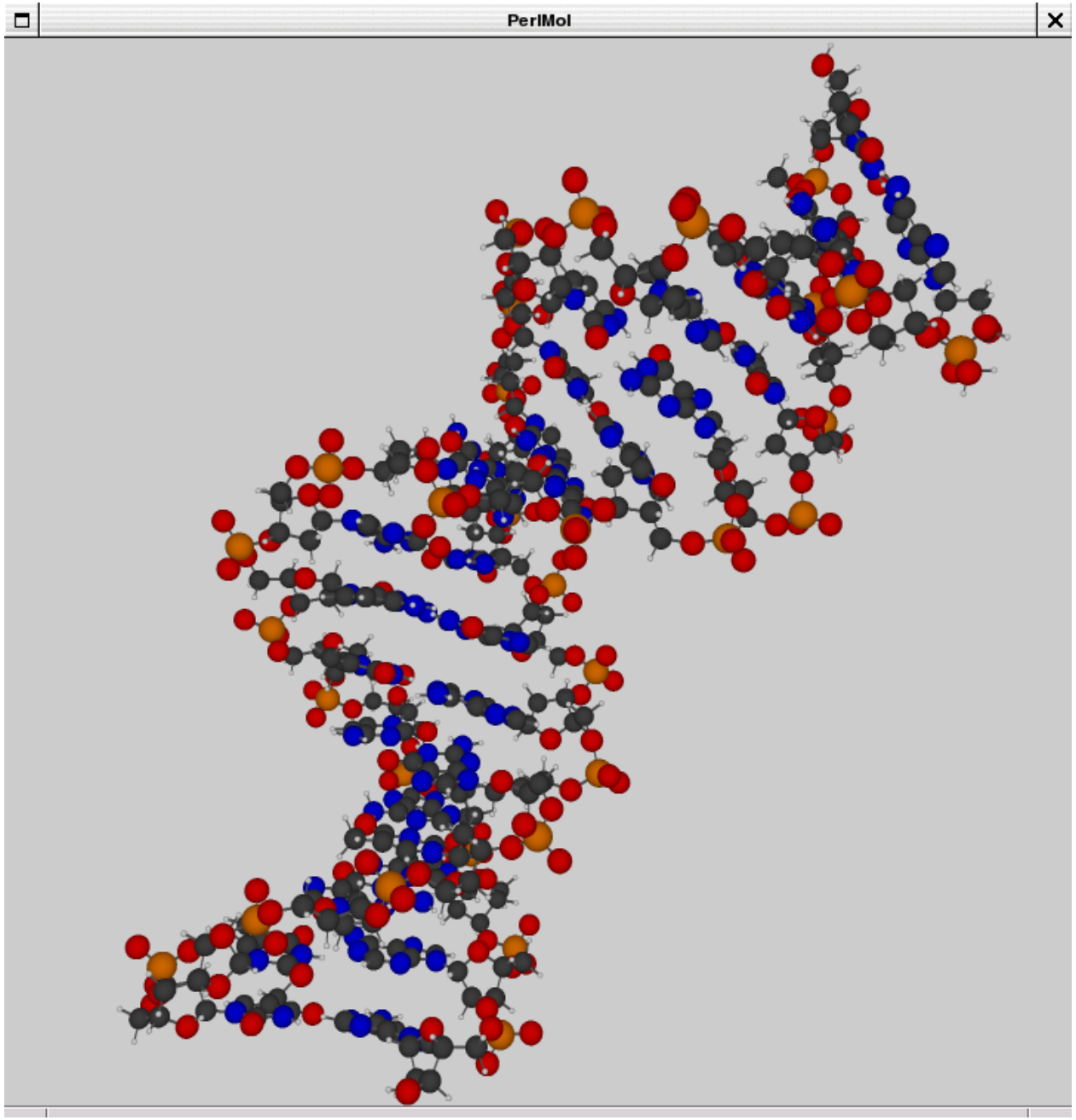


A 3D fractal

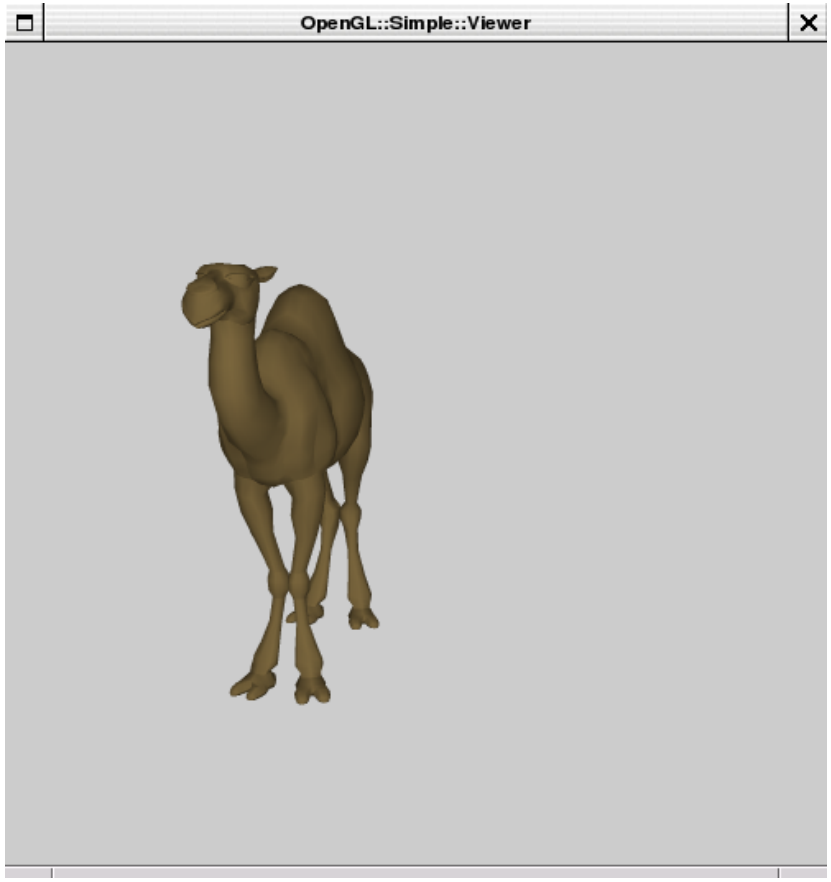
```
glBegin(GL_POINTS);
for (1..$npoints) {
  my $j = int(rand(4)); # 0 <= $j <= 3

  $x += 0.5*( $fixp[$j]->[0] - $x );
  $y += 0.5*( $fixp[$j]->[1] - $y );
  $z += 0.5*( $fixp[$j]->[2] - $z );

  $cr+= 0.5*( $col[$j]->[0] - $cr );
  $cg+= 0.5*( $col[$j]->[1] - $cg );
  $cb+= 0.5*( $col[$j]->[2] - $cb );

  glColor($cr,$cg,$cb);
  glVertex($x,$y,$z);
}
glEnd;
```



```
use Chemistry::File::PDB;
use Chemistry::Bond::Find ':all';
..
my $mol = Chemistry::MacroMol->read('dna.pdb;);
..
sub make_model {
    my $i = 0;
    recenter();
    my @atoms = $mol->atoms;
    for my $atom ( @atoms ) {
        my $mass   = log( 1 + $atom->mass ) / $mass_scale;
        my $color  = $element_colours{ $atom->symbol } || $colour{cyan};

        my @coords = $atom->coords->array;
        push @ballpoints, [ $color, $mass, @coords ];
    }
    for my $bond ( $mol->bonds ) {
        my ( $from, $to ) = $bond->atoms;
        my @from = $from->coords->array;
        my @to   = $to->coords->array;
        push @ballsticks, [ \@from, \@to ];
    }
}
```

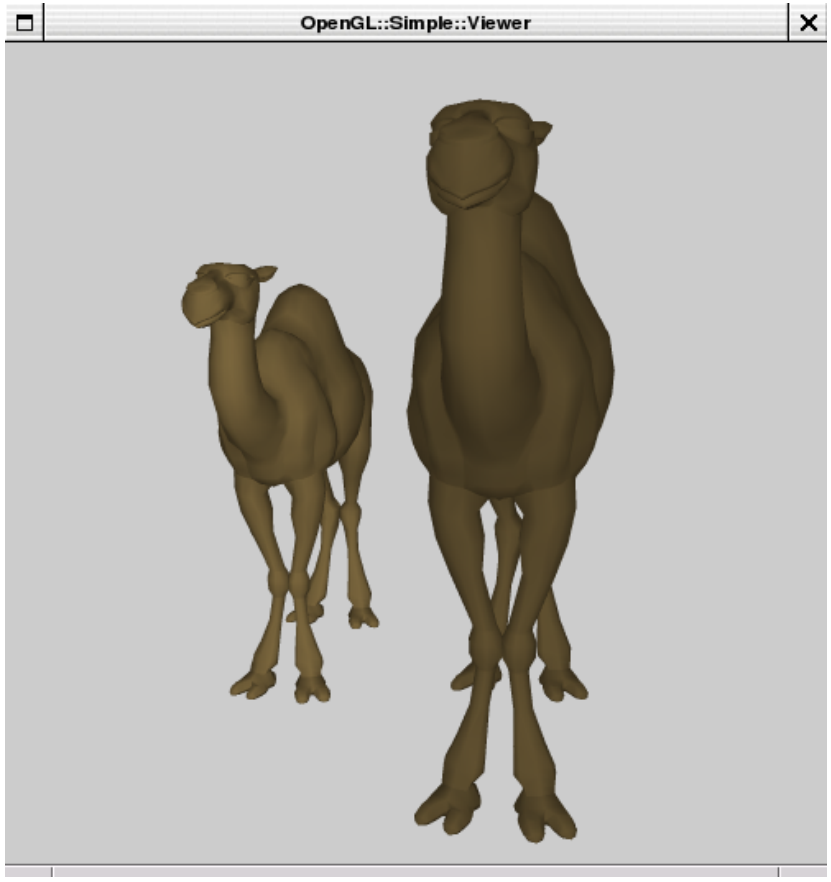

PerlMol

```
use OpenGL::Simple ':all';
use OpenGL::Simple::GLUT ':all';
use OpenGL::Simple::Viewer;
use OpenGL::GLM;

glutInit;
my ($model,$list);

my $v = new OpenGL::Simple::Viewer(
  draw_geometry => sub {
    unless ($model) {
      $model = new OpenGL::GLM('camel.obj');
      $model->Unitize;
      $list = $model->List(GLM_SMOOTH);
    }
    glColor(0.6,0.5,0.3);
    glCallList($list);
  },
);
glutMainLoop;
```

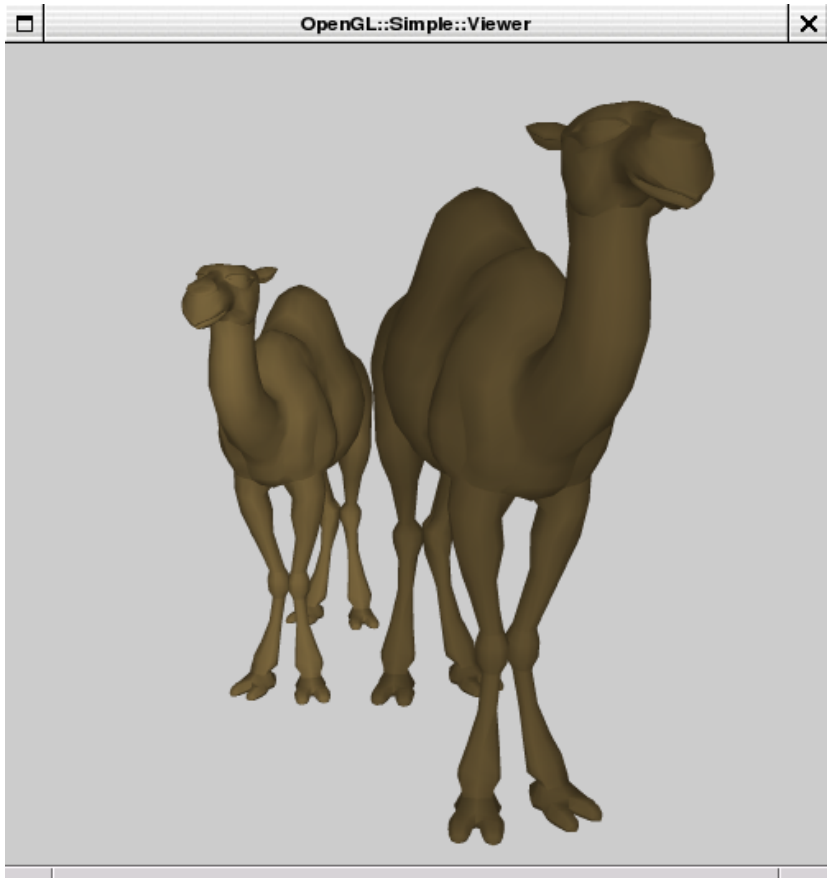


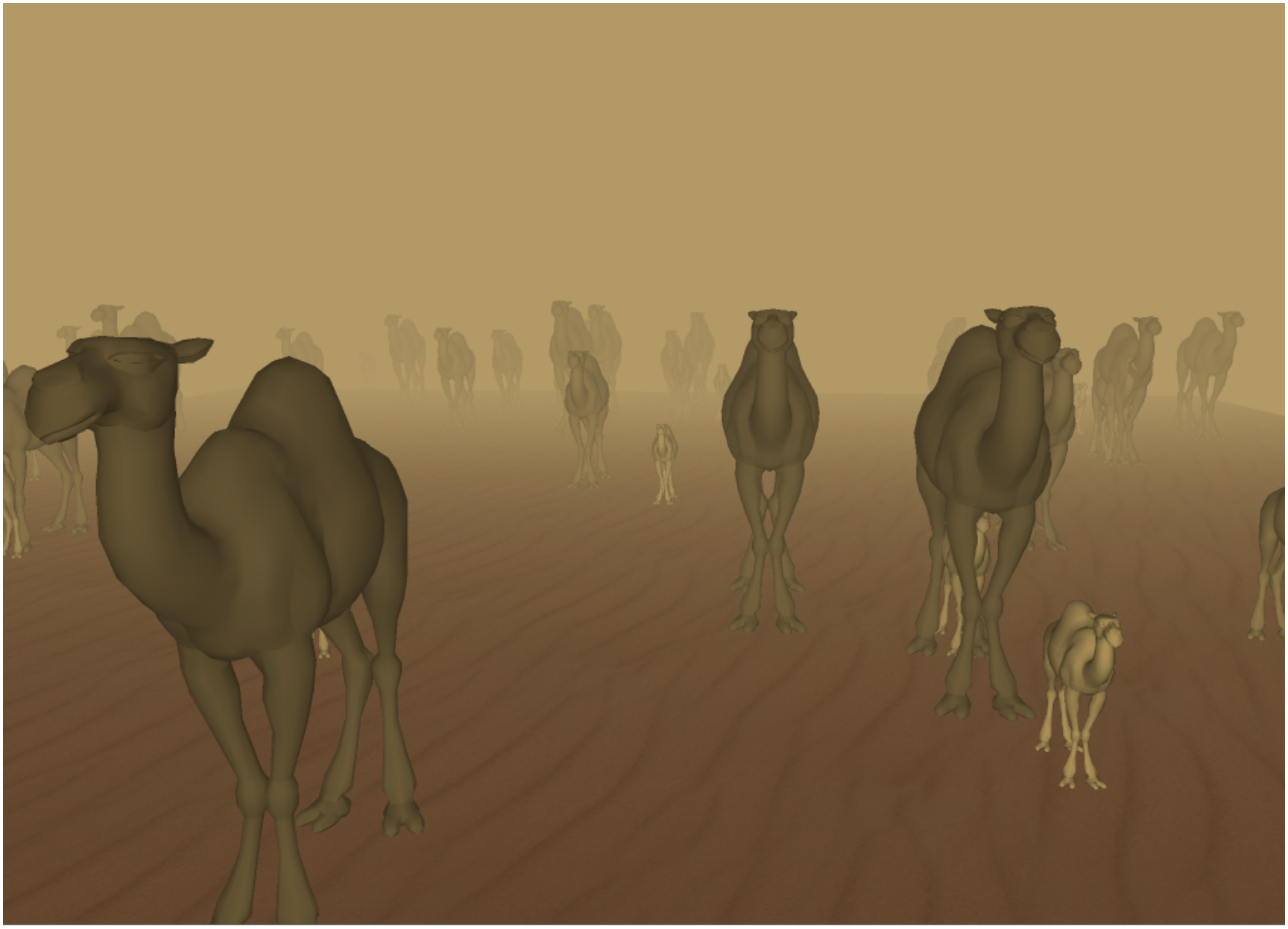
```
glCallList($list);
glTranslate(1,0,0);
glCallList($list);
```

```
glCallList($list);
glTranslate(1,0,0);
glScale(1.5,1.5,1.5);
glCallList($list);
```



```
glCallList($list);
glTranslate(1,0,0);
glScale(1.5,1.5,1.5);
glRotate(25,0,1,0);
glCallList($list);
```





Questions?