

CS-4470
Image Synthesis

Part 4

The Accumulation Buffer

Notes and Programs by:

Dr. Michael J. Zyda

Multi-Pass Anti-Aliasing with the Accumulation Buffer 3
Clearing the Accumulation Buffer 4
The glAccum() Command and the Accumulation Buffer 5
Scene Antialiasing with the Accumulation Buffer 6
accpersp.c 7
jitter.h 12
Additional Notes on the Accumulation Buffer 16
Motion Blur 17
Depth of Field 18
dof.c 19

Multi-Pass Anti-Aliasing with the Accumulation Buffer

-- Some time ago, we talked about techniques for computing pixel area coverage for various primitives in a one-pass method for anti-aliasing.

-- One thing that one discovers is that when it comes down to anti-aliasing polygons that we end up having to sort the polygons and that z-buffering is no longer good enough.

-- Now the problem with sorting and splitting polygons is again the problem that we are then out of the realm of real-time, interactive 3D graphics.

-- There is an elegant alternative approach called *accumulation* that anti-aliases points, lines and polygons and allows usage of the z-buffer for hidden surface elimination.

-- This technique, though, requires multiple passes at drawing the display data.

-- The technique also supports the following advanced rendering techniques:

-- Motion blur

-- Depth of field

-- Soft shadows

-- Accumulation is somewhat like blending in that multiple images are composited to produce the final image.

-- It differs from blending in that its operation is completely separated from the rendering of a single frame.

-- The accumulation buffer is an extended range bitplane bank the same size as the normal framebuffer.

-- You do not draw images into the accumulation buffer.

-- Rather, images drawn into the front or back buffer of the normal frame buffer are added to the contents of the accumulation buffer after they have been completely rendered.

-- When the accumulation is finished, the result is copied back into a color buffer for viewing.

Clearing the Accumulation Buffer

-- We clear the accumulation buffer in a manner similar to clearing the frame and z-buffer:

```
// Set the color we should clear the accumulation buffer to ...  
glClearAccum(GLfloat red, GLfloat green, GLfloat blue, GLfloat alpha);
```

```
// Clear the accumulation buffer.  
glClear(GL_ACCUM_BUFFER_BIT);
```

Note: With the accumulation buffer, we also have the option of not clearing it and loading it the first time we use it.

The glAccum() Command and the Accumulation Buffer

```
glAccum(GLenum op, GLfloat value);
```

Controls the accumulation buffer. The *op* parameter selects the operation, and *value* is a number to be used in that operation. The possible operations are: GL_ACCUM, GL_LOAD, GL_RETURN, GL_ADD, and GL_MULT.

GL_LOAD reads each pixel from the buffer currently selected for reading with glReadBuffer(), multiplies the RGBA values by *value*, and places those modified RGBA values into the accumulation buffer. The first use of the accumulation buffer typically starts this way.

```
glReadBuffer(GLenum mode);
```

mode can be GL_FRONT, and GL_BACK (and many other choices). By default, *mode* is GL_FRONT in single-buffered configurations, and GL_BACK in double-buffered configurations.

GL_ACCUM reads each pixel from the buffer currently selected for reading with glReadBuffer(), multiplies the RGBA values by *value*, and adds the result to the accumulation buffer.

GL_RETURN takes values from the accumulation buffer, multiplies them by *value*, and places the result in the color buffer(s) enabled for writing.

GL_ADD and GL_MULT simply add or multiply the value of each pixel in the accumulation buffer by *value*, and then return it to the accumulation buffer. For GL_MULT, *value* is clamped to be in the range [-1.0, 1.0]. For GL_ADD, no clamping occurs.

Scene Antialiasing with the Accumulation Buffer

Steps

1. Clear the accumulation buffer.

```
glClearAccum(0.0, 0.0, 0.0, 0.0); // Clear color is 0.0
glClear(GL_ACCUM_BUFFER_BIT);
```

2. Loop several times (ntimes) through code that draws the image in a slightly different position, accumulating the data with `glAccum(GL_ACCUM, 1.0/ntimes)`;

```
GLint viewport[4];

glGetIntegerv(GL_VIEWPORT, viewport);

ntimes = 8; // Assume that we draw the scene 8 times for each antialiased image.

for(jitter=0; jitter < ntimes; jitter++)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    accPerspective(50.0,
                  (GLdouble) viewport[2]/(GLdouble) viewport[3], // aspect ratio
                  1.0, 15.0, // distance to near and far planes.
                  j8[jitter].x, j8[jitter].y, // The x and y jitter values
                  0.0, 0.0, 1.0); // These last three values are used for depth of field.

    displayObjects();
    glAccum(GL_ACCUM, 1.0/ntimes);
}
```

Note: `accPerspective()` is used in place of `gluPerspective()`. The first four parameters are the same. The 5th & 6th are for the jitter amounts. The last three are for depth of field effects and 0.0, 0.0, 1.0 is the “don’t do anything” useful for scene antialiasing.

3. Finally call `glAccum(GL_RETURN, 1.0)`; to get the final picture into a color buffer.

```
glAccum(GL_RETURN, 1.0);
```

accpersp.c

```
/* accpersp.c
 */
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include "aux.h"
#include "jitter.h"

#define PI_ 3.14159265358979323846

/*          accFrustum()
 * The first 6 arguments are identical to the glFrustum() call.
 *
 * pixdx and pixdy are anti-alias jitter in pixels.
 * Set both equal to 0.0 for no anti-alias jitter.
 * eyedx and eyedy are depth-of field jitter in pixels.
 * Set both equal to 0.0 for no depth of field effects.
 *
 * focus is distance from eye to plane in focus.
 * focus must be greater than, but not equal to 0.0.
 *
 * Note that accFrustum() calls glTranslatef(). You will
 * probably want to insure that your ModelView matrix has been
 * initialized to identity before calling accFrustum().
 */
void accFrustum(GLdouble left, GLdouble right, GLdouble bottom, GLdouble top,
  GLdouble near, GLdouble far, GLdouble pixdx, GLdouble pixdy,
  GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
  GLdouble xwsize, ywsize;
  GLdouble dx, dy;
  GLint viewport[4];

  glGetIntegerv (GL_VIEWPORT, viewport);

  xwsize = right - left;
  ywsize = top - bottom;

  dx = -(pixdx*xwsize/(GLdouble) viewport[2] + eyedx*near/focus);
  dy = -(pixdy*ywsize/(GLdouble) viewport[3] + eyedy*near/focus);

  glMatrixMode(GL_PROJECTION);
  glLoadIdentity();
  glFrustum (left + dx, right + dx, bottom + dy, top + dy, near, far);
  glMatrixMode(GL_MODELVIEW);
  glLoadIdentity();
  glTranslatef (-eyedx, -eyedy, 0.0);
}
```

```

/*  accPerspective()
*
*  The first 4 arguments are identical to the gluPerspective() call.
*  pixdx and pixdy are anti-alias jitter in pixels.
*  Set both equal to 0.0 for no anti-alias jitter.
*  eyedx and eyedy are depth-of field jitter in pixels.
*  Set both equal to 0.0 for no depth of field effects.
*
*  focus is distance from eye to plane in focus.
*  focus must be greater than, but not equal to 0.0.
*
*  Note that accPerspective() calls accFrustum().
*/
void accPerspective(GLdouble fovy, GLdouble aspect,
    GLdouble near, GLdouble far, GLdouble pixdx, GLdouble pixdy,
    GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
    GLdouble fov2, left, right, bottom, top;

    fov2 = ((fovy*PI_) / 180.0) / 2.0;

    top = near / (cos(fov2) / sin(fov2));
    bottom = -top;

    right = top * aspect;
    left = -right;

    accFrustum (left, right, bottom, top, near, far,
                pixdx, pixdy, eyedx, eyedy, focus);
}

```

```

/* Initialize lighting and other values.
 */
void myinit(void)
{
    GLfloat mat_ambient[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat mat_specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat light_position[] = { 0.0, 0.0, 10.0, 1.0 };
    GLfloat lm_ambient[] = { 0.2, 0.2, 0.2, 1.0 };

    glMaterialfv(GL_FRONT, GL_AMBIENT, mat_ambient);
    glMaterialfv(GL_FRONT, GL_SPECULAR, mat_specular);
    glMaterialf(GL_FRONT, GL_SHININESS, 50.0);
    glLightfv(GL_LIGHT0, GL_POSITION, light_position);
    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lm_ambient);

    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);
    glShadeModel (GL_FLAT);

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClearAccum(0.0, 0.0, 0.0, 0.0);
}

```

```

void displayObjects(void)
{
    GLfloat torus_diffuse[] = { 0.7, 0.7, 0.0, 1.0 };
    GLfloat cube_diffuse[] = { 0.0, 0.7, 0.7, 1.0 };
    GLfloat sphere_diffuse[] = { 0.7, 0.0, 0.7, 1.0 };
    GLfloat octa_diffuse[] = { 0.7, 0.4, 0.4, 1.0 };

    glPushMatrix ();
    glTranslatef (0.0, 0.0, -5.0);
    glRotatef (30.0, 1.0, 0.0, 0.0);

    glPushMatrix ();
    glTranslatef (-0.80, 0.35, 0.0);
    glRotatef (100.0, 1.0, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, torus_diffuse);
    auxSolidTorus (0.275, 0.85);
    glPopMatrix ();

    glPushMatrix ();
    glTranslatef (-0.75, -0.50, 0.0);
    glRotatef (45.0, 0.0, 0.0, 1.0);
    glRotatef (45.0, 1.0, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, cube_diffuse);
    auxSolidCube (1.5);
    glPopMatrix ();

    glPushMatrix ();
    glTranslatef (0.75, 0.60, 0.0);
    glRotatef (30.0, 1.0, 0.0, 0.0);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, sphere_diffuse);
    auxSolidSphere (1.0);
    glPopMatrix ();

    glPushMatrix ();
    glTranslatef (0.70, -0.90, 0.25);
    glMaterialfv(GL_FRONT, GL_DIFFUSE, octa_diffuse);
    auxSolidOctahedron (1.0);
    glPopMatrix ();

    glPopMatrix ();
}

```

```

#define ACSIZE          8

void display(void)
{
    GLint viewport[4];
    int jitter;

    glGetIntegerv (GL_VIEWPORT, viewport);

    glClear(GL_ACCUM_BUFFER_BIT);
    for (jitter = 0; jitter < ACSIZE; jitter++) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        accPerspective (50.0,
            (GLdouble) viewport[2]/(GLdouble) viewport[3],
            1.0, 15.0, j8[jitter].x, j8[jitter].y,
            0.0, 0.0, 1.0);
        displayObjects ();
        glAccum(GL_ACCUM, 1.0/ACSIZE);
    }
    glAccum (GL_RETURN, 1.0);
    glFlush();
}

void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
}

/* Main Loop
 * Open window with initial window size, title bar,
 * RGBA display mode, and handle input events.
 */
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE | AUX_RGB
        | AUX_ACCUM | AUX_DEPTH);
    auxInitPosition (0, 0, 250, 250);
    auxInitWindow (argv[0]);
    myinit();
    auxReshapeFunc (myReshape);
    auxMainLoop(display);
}

```

jitter.h

```
/*
jitter.h

This file contains jitter point arrays for 2,3,4,8,15,24 and 66 jitters.

The arrays are named j2, j3, etc. Each element in the array has the form,
for example, j8[0].x and j8[0].y

Values are floating point in the range  $-0.5 < x < 0.5$ ,  $-0.5 < y < 0.5$ , and
have a gaussian distribution around the origin.

Use these to do model jittering for scene anti-aliasing and view volume
jittering for depth of field effects. Use in conjunction with the
accwindow() routine.
*/

typedef struct
{
    GLfloat x, y;
} jitter_point;

#define MAX_SAMPLES 66

/* 2 jitter points */
jitter_point j2[] =
{
    { 0.246490, 0.249999},
    {-0.246490, -0.249999}
};

/* 3 jitter points */
jitter_point j3[] =
{
    {-0.373411, -0.250550},
    { 0.256263, 0.368119},
    { 0.117148, -0.117570}
};

/* 4 jitter points */
jitter_point j4[] =
{
    {-0.208147, 0.353730},
    { 0.203849, -0.353780},
    {-0.292626, -0.149945},
    { 0.296924, 0.149994}
};
```

```

/* 8 jitter points */
jitter_point j8[] =
{
    {-0.334818,  0.435331},
    { 0.286438, -0.393495},
    { 0.459462,  0.141540},
    {-0.414498, -0.192829},
    {-0.183790,  0.082102},
    {-0.079263, -0.317383},
    { 0.102254,  0.299133},
    { 0.164216, -0.054399}
};

```

```

/* 15 jitter points */
jitter_point j15[] =
{
    { 0.285561,  0.188437},
    { 0.360176, -0.065688},
    {-0.111751,  0.275019},
    {-0.055918, -0.215197},
    {-0.080231, -0.470965},
    { 0.138721,  0.409168},
    { 0.384120,  0.458500},
    {-0.454968,  0.134088},
    { 0.179271, -0.331196},
    {-0.307049, -0.364927},
    { 0.105354, -0.010099},
    {-0.154180,  0.021794},
    {-0.370135, -0.116425},
    { 0.451636, -0.300013},
    {-0.370610,  0.387504}
};

```

```

/* 24 jitter points */
jitter_point j24[] =
{
    { 0.030245,  0.136384},
    { 0.018865, -0.348867},
    {-0.350114, -0.472309},
    { 0.222181,  0.149524},
    {-0.393670, -0.266873},
    { 0.404568,  0.230436},
    { 0.098381,  0.465337},
    { 0.462671,  0.442116},
    { 0.400373, -0.212720},
    {-0.409988,  0.263345},
    {-0.115878, -0.001981},
    { 0.348425, -0.009237},
    {-0.464016,  0.066467},
    {-0.138674, -0.468006},
    { 0.144932, -0.022780},
    {-0.250195,  0.150161},

```

```

        {-0.181400, -0.264219},
        { 0.196097, -0.234139},
        {-0.311082, -0.078815},
        { 0.268379,  0.366778},
        {-0.040601,  0.327109},
        {-0.234392,  0.354659},
        {-0.003102, -0.154402},
        { 0.297997, -0.417965}
};

```

```

/* 66 jitter points */
jitter_point j66[] =
{

```

```

    { 0.266377, -0.218171},
    {-0.170919, -0.429368},
    { 0.047356, -0.387135},
    {-0.430063,  0.363413},
    {-0.221638, -0.313768},
    { 0.124758, -0.197109},
    {-0.400021,  0.482195},
    { 0.247882,  0.152010},
    {-0.286709, -0.470214},
    {-0.426790,  0.004977},
    {-0.361249, -0.104549},
    {-0.040643,  0.123453},
    {-0.189296,  0.438963},
    {-0.453521, -0.299889},
    { 0.408216, -0.457699},
    { 0.328973, -0.101914},
    {-0.055540, -0.477952},
    { 0.194421,  0.453510},
    { 0.404051,  0.224974},
    { 0.310136,  0.419700},
    {-0.021743,  0.403898},
    {-0.466210,  0.248839},
    { 0.341369,  0.081490},
    { 0.124156, -0.016859},
    {-0.461321, -0.176661},
    { 0.013210,  0.234401},
    { 0.174258, -0.311854},
    { 0.294061,  0.263364},
    {-0.114836,  0.328189},
    { 0.041206, -0.106205},
    { 0.079227,  0.345021},
    {-0.109319, -0.242380},
    { 0.425005, -0.332397},
    { 0.009146,  0.015098},
    {-0.339084, -0.355707},
    {-0.224596, -0.189548},
    { 0.083475,  0.117028},
    { 0.295962, -0.334699},
    { 0.452998,  0.025397},
    { 0.206511, -0.104668},

```

{ 0.447544, -0.096004},
{-0.108006, -0.002471},
{-0.380810, 0.130036},
{-0.242440, 0.186934},
{-0.200363, 0.070863},
{-0.344844, -0.230814},
{ 0.408660, 0.345826},
{-0.233016, 0.305203},
{ 0.158475, -0.430762},
{ 0.486972, 0.139163},
{-0.301610, 0.009319},
{ 0.282245, -0.458671},
{ 0.482046, 0.443890},
{-0.121527, 0.210223},
{-0.477606, -0.424878},
{-0.083941, -0.121440},
{-0.345773, 0.253779},
{ 0.234646, 0.034549},
{ 0.394102, -0.210901},
{-0.312571, 0.397656},
{ 0.200906, 0.333293},
{ 0.018703, -0.261792},
{-0.209349, -0.065383},
{ 0.076248, 0.478538},
{-0.073036, -0.355064},
{ 0.145087, 0.221726}

};

Additional Notes on the Accumulation Buffer

-- In many applications, the use of the accumulation buffer can be best conditioned on user input.

-- For example, one traditional way of handling real-time movement is to defer image quality until the viewpoint is stopped.

-- This means that we draw the picture without anti-aliasing while we are moving our viewpoint and only turn on anti-aliasing (accumulation buffering) when our viewpoint is stopped.

-- Anti-aliasing filter quality improves when the samples are distributed in a circular pattern that is larger than a pixel.

-- The filter quality can further be improved by shaping it as a symmetric Gaussian function,

-- Either by changing the density of sample locations within the circle

or

-- By keeping the sample density constant and assigning different weights to the samples.

-- A circularly symmetric Gaussian filter function yields smoother edges than does a unit-size box filter.

-- Regardless of the filter function, fewer samples are required to achieve a given anti-aliasing quality level when the samples are distributed in a random fashion, rather than in regular rows and columns.

Motion Blur

-- Suppose your scene has some stationary and some moving objects in it, and you want to make a motion-blurred image extending over a small interval of time.

-- Set up the accumulation buffer in the same way, but instead of spatially jittering the images, jitter them temporally.

-- This means we draw the scenes at slightly later times, with the moving objects then at different locations.

-- The entire scene can be made successively dimmer calling:

```
glAccum(GL_MULT, decayFactor);
```

as the scene is drawn into the accumulation buffer, where `decayFactor` is a number between 0.0 and 1.0.

Smaller numbers for `decayFactor` cause the object to appear to be moving faster.

We transfer the completed picture with the object's current position and "vapor trail" of previous positions from the accumulation buffer to the color buffer with:

```
glAccum(GL_RETURN, 1.0);
```

The image looks correct even if items move at different speeds, or if some of them are accelerated.

Depth of Field

- A photograph made with a camera is in perfect focus only for items lying on a single plane a certain distance from the film.
- The farther an item is from this plane, the more out of focus it is.
- The depth of field for a camera is a region about the plane of perfect focus where items are out of focus by a small enough amount.
- Under normal conditions, everything you draw is in perfect focus. The accumulation buffer can be used to approximate what you would see in a photograph where items are more and more blurred as their distance from a plane of perfect focus increases.
- To achieve this result, draw the scene repeatedly using calls with different argument values to `glFrustum()`. Choose the arguments so that the position of the viewpoint varies slightly around its true position and so that each frustum shares a common rectangle that lies in the plane of perfect focus. The results of the renderings should be averaged in the usual way using the accumulation buffer.
- As you recall with scene antialiasing, the 5th & 6th parameters of `accPerspective()` jitter the viewing volumes in the x and y directions. For the depth of field effect, you want to jitter the volume while holding it stationary at the focal plane. The focal plane is the depth value defined by the ninth (last) parameter to `accPerspective()`, which is $z=0$ in the following sample program. The amount of blur is determined by multiplying the x and y jitter values (7th and 8th parameters) by a constant. The constant is determined by experimentation.

Note: the 5th and 6th parameters in the example are 0.0 so scene antialiasing is turned off.

dof.c

```
/*
 * dof.c
 * This program demonstrates use of the accumulation buffer to
 * create an out-of-focus depth-of-field effect.  The teapots
 * are drawn several times into the accumulation buffer.  The
 * viewing volume is jittered, except at the focal point, where
 * the viewing volume is at the same position, each time.  In
 * this case, the gold teapot remains in focus.
 */
#include <GL/gl.h>
#include <GL/glu.h>
#include <math.h>
#include "aux.h"
#include "jitter.h"

#define PI_ 3.14159265358979323846

/*          accFrustum()
 * The first 6 arguments are identical to the glFrustum() call.
 *
 * pixdx and pixdy are anti-alias jitter in pixels.
 * Set both equal to 0.0 for no anti-alias jitter.
 * eyedx and eyedy are depth-of field jitter in pixels.
 * Set both equal to 0.0 for no depth of field effects.
 *
 * focus is distance from eye to plane in focus.
 * focus must be greater than, but not equal to 0.0.
 *
 * Note that accFrustum() calls glTranslatef().  You will
 * probably want to insure that your ModelView matrix has been
 * initialized to identity before calling accFrustum().
 */
void accFrustum(GLdouble left, GLdouble right, GLdouble bottom,
                GLdouble top, GLdouble near, GLdouble far, GLdouble pixdx,
                GLdouble pixdy, GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
    GLdouble xysize, yysize;
    GLdouble dx, dy;
    GLint viewport[4];

    glGetIntegerv (GL_VIEWPORT, viewport);

    xysize = right - left;
    yysize = top - bottom;

    dx = -(pixdx*xysize/(GLdouble) viewport[2] + eyedx*near/focus);
    dy = -(pixdy*yysize/(GLdouble) viewport[3] + eyedy*near/focus);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum (left + dx, right + dx, bottom + dy, top + dy, near, far);
    glMatrixMode(GL_MODELVIEW);
}
```

```

    glLoadIdentity();
    glTranslatef (-eyedx, -eyedy, 0.0);
}

/* accPerspective()
 *
 * The first 4 arguments are identical to the gluPerspective() call.
 * pixdx and pixdy are anti-alias jitter in pixels.
 * Set both equal to 0.0 for no anti-alias jitter.
 * eyedx and eyedy are depth-of field jitter in pixels.
 * Set both equal to 0.0 for no depth of field effects.
 *
 * focus is distance from eye to plane in focus.
 * focus must be greater than, but not equal to 0.0.
 *
 * Note that accPerspective() calls accFrustum().
 */
void accPerspective(GLdouble fovy, GLdouble aspect,
    GLdouble near, GLdouble far, GLdouble pixdx, GLdouble pixdy,
    GLdouble eyedx, GLdouble eyedy, GLdouble focus)
{
    GLdouble fov2, left, right, bottom, top;

    fov2 = ((fovy*PI_) / 180.0) / 2.0;

    top = near / (cos(fov2) / sin(fov2));
    bottom = -top;

    right = top * aspect;
    left = -right;

    accFrustum (left, right, bottom, top, near, far,
                pixdx, pixdy, eyedx, eyedy, focus);
}

```

```

void myinit(void)
{
    GLfloat ambient[] = { 0.0, 0.0, 0.0, 1.0 };
    GLfloat diffuse[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat specular[] = { 1.0, 1.0, 1.0, 1.0 };
    GLfloat position[] = { 0.0, 3.0, 3.0, 0.0 };

    GLfloat lmodel_ambient[] = { 0.2, 0.2, 0.2, 1.0 };
    GLfloat local_view[] = { 0.0 };

    glEnable(GL_DEPTH_TEST);
    glDepthFunc(GL_LESS);

    glLightfv(GL_LIGHT0, GL_AMBIENT, ambient);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, diffuse);
    glLightfv(GL_LIGHT0, GL_POSITION, position);

    glLightModelfv(GL_LIGHT_MODEL_AMBIENT, lmodel_ambient);
    glLightModelfv(GL_LIGHT_MODEL_LOCAL_VIEWER, local_view);

    glFrontFace (GL_CW);
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_AUTO_NORMAL);
    glEnable(GL_NORMALIZE);

    glMatrixMode (GL_MODELVIEW);
    glLoadIdentity ();

    glClearColor(0.0, 0.0, 0.0, 0.0);
    glClearAccum(0.0, 0.0, 0.0, 0.0);
}

```

```
void renderTeapot (GLfloat x, GLfloat y, GLfloat z,
  GLfloat ambr, GLfloat ambg, GLfloat ambb,
  GLfloat difr, GLfloat difg, GLfloat difb,
  GLfloat specr, GLfloat specg, GLfloat specb, GLfloat shine)
{
  float mat[3];

  glPushMatrix();
  glTranslatef (x, y, z);
  mat[0] = ambr; mat[1] = ambg; mat[2] = ambb;
  glMaterialfv (GL_FRONT, GL_AMBIENT, mat);
  mat[0] = difr; mat[1] = difg; mat[2] = difb;
  glMaterialfv (GL_FRONT, GL_DIFFUSE, mat);
  mat[0] = specr; mat[1] = specg; mat[2] = specb;
  glMaterialfv (GL_FRONT, GL_SPECULAR, mat);
  glMaterialf (GL_FRONT, GL_SHININESS, shine*128.0);
  auxSolidTeapot(0.5);
  glPopMatrix();
}
```

```

/* display() draws 5 teapots into the accumulation buffer
 * several times; each time with a jittered perspective.
 * The focal point is at z = 5.0, so the gold teapot will
 * stay in focus. The amount of jitter is adjusted by the
 * magnitude of the accPerspective() jitter; in this example, 0.33.
 * In this example, the teapots are drawn 8 times. See jitter.h
 */
void display(void)
{
    int jitter;
    GLint viewport[4];

    glGetIntegerv (GL_VIEWPORT, viewport);
    glClear(GL_ACCUM_BUFFER_BIT);

    for (jitter = 0; jitter < 8; jitter++) {
        glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
        accPerspective (45.0,
            (GLdouble) viewport[2]/(GLdouble) viewport[3],
            1.0, 15.0, 0.0, 0.0,
            0.33*j8[jitter].x, 0.33*j8[jitter].y, 5.0);
/*
        ruby, gold, silver, emerald, and cyan teapots*/

renderTeapot (-1.1, -0.5, -4.5, 0.1745, 0.01175, 0.01175,
0.61424, 0.04136, 0.04136, 0.727811, 0.626959, 0.626959, 0.6);
renderTeapot (-0.5, -0.5, -5.0, 0.24725, 0.1995, 0.0745,
0.75164, 0.60648, 0.22648, 0.628281, 0.555802, 0.366065, 0.4);
renderTeapot (0.2, -0.5, -5.5, 0.19225, 0.19225, 0.19225,
0.50754, 0.50754, 0.50754, 0.508273, 0.508273, 0.508273, 0.4);
renderTeapot (1.0, -0.5, -6.0, 0.0215, 0.1745, 0.0215,
0.07568, 0.61424, 0.07568, 0.633, 0.727811, 0.633, 0.6);
renderTeapot (1.8, -0.5, -6.5, 0.0, 0.1, 0.06, 0.0, 0.50980392,
0.50980392, 0.50196078, 0.50196078, 0.50196078, .25);

        glAccum (GL_ACCUM, 0.125);
    }

    glAccum (GL_RETURN, 1.0);
    glFlush();
}

void myReshape(GLsizei w, GLsizei h)
{
    glViewport(0, 0, w, h);
}

```

```
/* Main Loop
 * Open window with initial window size, title bar,
 * RGBA display mode, depth buffer, and handle input events.
 */
int main(int argc, char** argv)
{
    auxInitDisplayMode (AUX_SINGLE | AUX_RGB
                       | AUX_ACCUM | AUX_DEPTH);
    auxInitPosition (0, 0, 400, 400);
    auxInitWindow (argv[0]);
    myinit();
    auxReshapeFunc (myReshape);
    auxMainLoop(display);
}
```