Typical OpenGL/GLUT Main Program

```
#include <GL/glut.h>                        // GLUT, GLU, and OpenGL defs
int main(int argc, char** argv)             // program arguments
{
    glutInit(&argc, argv);                  // initialize glut and gl
                                            // double buffering and RGB
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGBA);
    glutInitWindowSize(400, 300);           // initial window size
    glutInitWindowPosition(0, 0);           // initial window position
    glutCreateWindow(argv[0]);              // create window

    ...initialize callbacks here (described below)...

    myInit();                               // your own initializations
    glutMainLoop();                         // turn control over to glut
    return 0; // we never return here; this just keeps the compiler happy
}
```

Display Mode	Meaning
GLUT_RGB	Use RGB colors
GLUT_RGBA	Use RGB plus α (recommended)
GLUT_INDEX	Use colormapped colors (not recommended)
GLUT_DOUBLE	Use double buffering (recommended)
GLUT_SINGLE	Use single buffering (not recommended)
GLUT_DEPTH	Use depth buffer (needed for hidden surface removal)

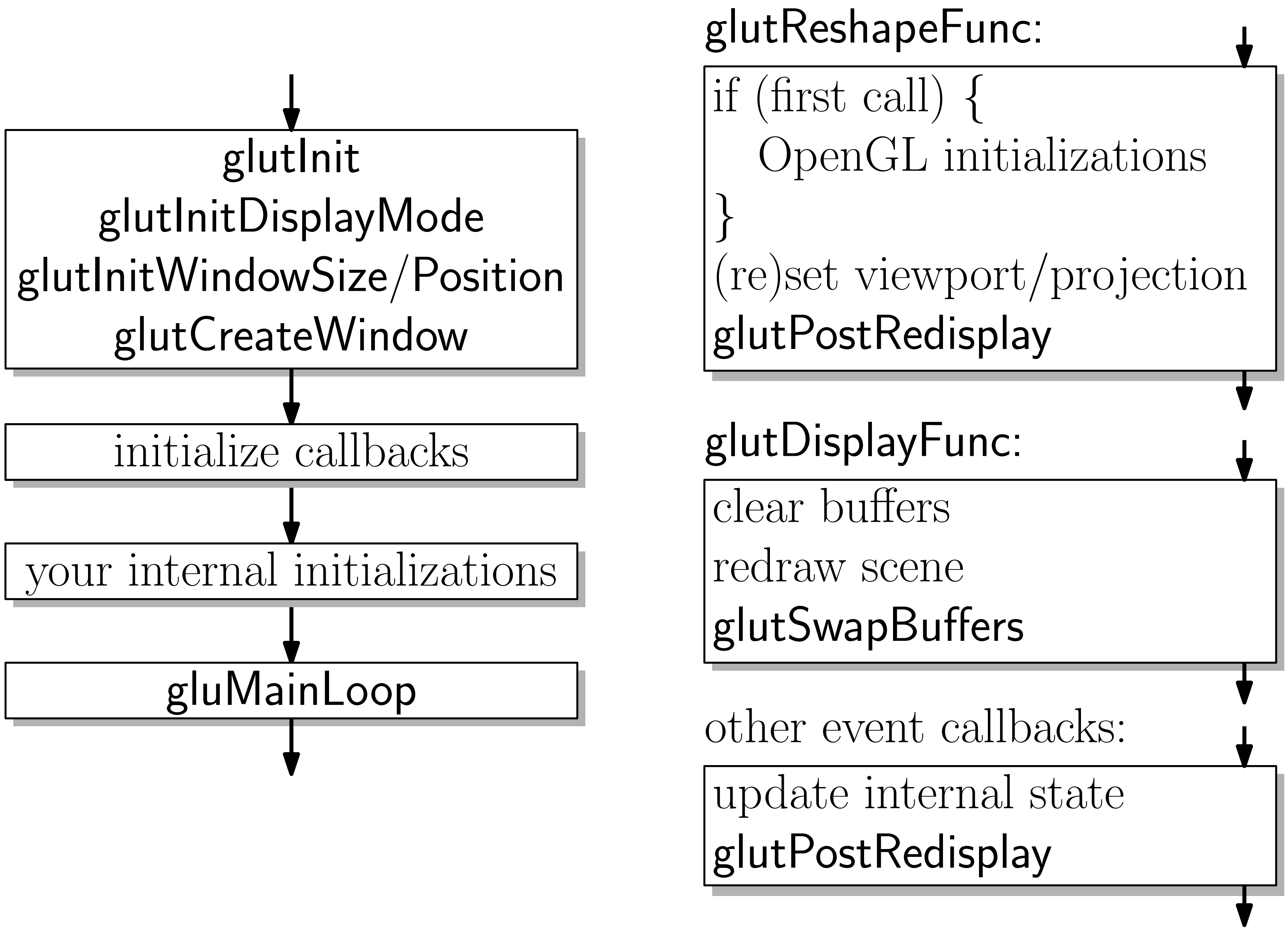


Fig. 9: General structure of an OpenGL program using GLUT.

Input Event	Callback request	User callback function prototype (return void)
Mouse button	glutMouseFunc	myMouse(int b, int s, int x, int y)
Mouse motion	glutPassiveMotionFunc	myMotion(int x, int y)
Keyboard key	glutKeyboardFunc	myKeyboard(unsigned char c, int x, int y)
System Event	Callback request	User callback function prototype (return void)
(Re)display	glutDisplayFunc	myDisplay()
(Re)size window	glutReshapeFunc	myReshape(int w, int h)
Timer event	glutTimerFunc	myTimer(int id)
Idle event	glutIdleFunc	myIdle()

Table 2: Common callbacks and the associated registration functions.

Typical Callback Setup

```
int main(int argc, char** argv)
{
    ...
    glutDisplayFunc(myDraw);                // set up the callbacks
    glutReshapeFunc(myReshape);
    glutMouseFunc(myMouse);
    glutKeyboardFunc(myKeyboard);
    glutTimerFunc(20, myTimeOut, 0);        // timer in 20/1000 seconds
    ...
}
```

Examples of Callback Functions for System Events

```
void myDraw() {                             // called to display window
   // ...insert your drawing code here ...
}
void myReshape(int w, int h) {              // called if reshaped
    windowWidth = w;                        // save new window size
    windowHeight = h;
    // ...may need to update the projection ...
    glutPostRedisplay();                    // request window redisplay
}
void myTimeOut(int id) {                    // called if timer event
    // ...advance the state of animation incrementally...
    glutPostRedisplay();                    // request redisplay
    glutTimerFunc(20, myTimeOut, 0);        // schedule next timer event
}
```

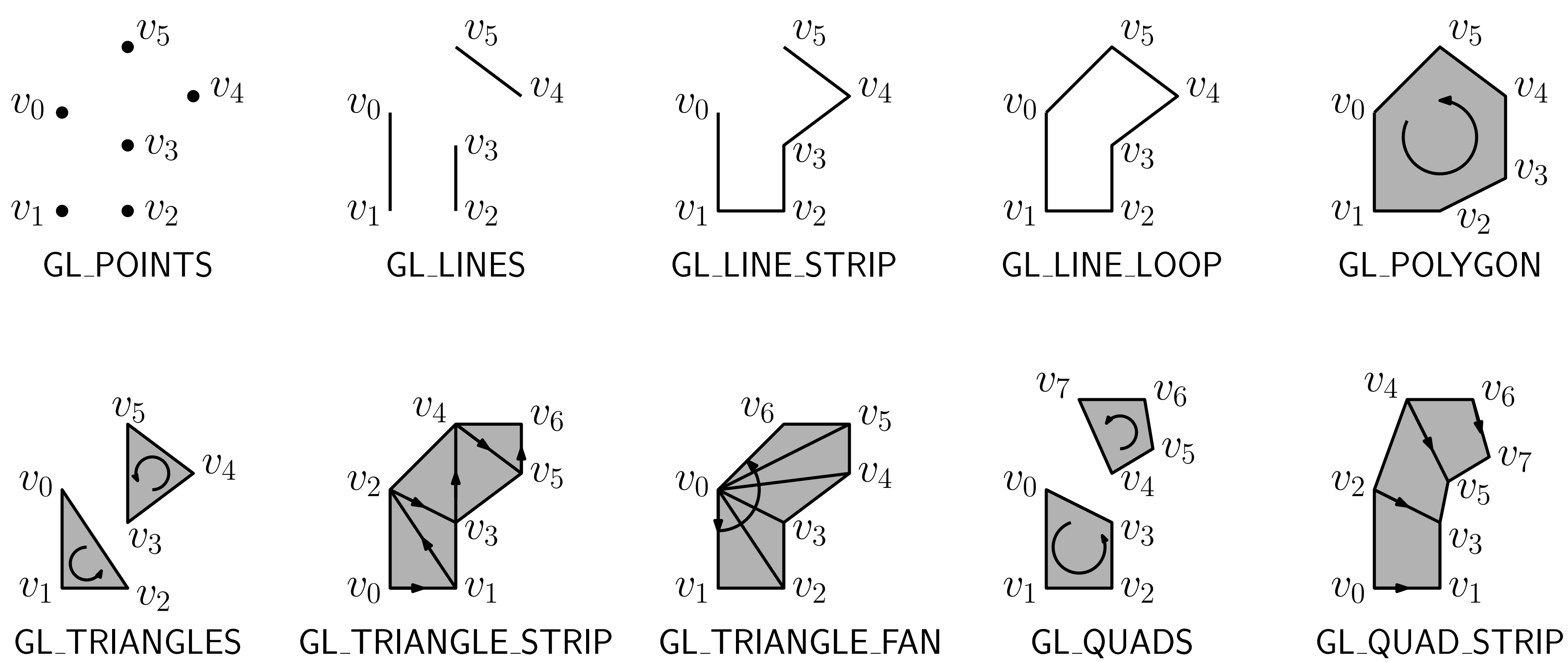
```
                                        // called if mouse click
void myMouse(int b, int s, int x, int y) {
    switch (b) {                        // b indicates the button
        case GLUT_LEFT_BUTTON:
            if (s == GLUT_DOWN)         // button pressed
                // ...
            else if (s == GLUT_UP)      // button released
                // ...
            break;
        // ...                         // other button events
    }
}
                                        // called if keyboard key hit
void myKeyboard(unsigned char c, int x, int y) {
   switch (c) {                         // c is the key that is hit
      case 'q':                         // 'q' means quit
          exit(0);
          break;
      // ...                           // other keyboard events
   }
}
```

GLUT Parameter Name	Meaning
GLUT_LEFT_BUTTON	left mouse button
GLUT_MIDDLE_BUTTON	middle mouse button
GLUT_RIGHT_BUTTON	right mouse button
GLUT_DOWN	mouse button pressed down
GLUT_UP	mouse button released

Sample Display Function

```
void myDisplay() {                              // display function
    glClear(GL_COLOR_BUFFER_BIT);               // clear the window

    glColor3f(1.0, 0.0, 0.0);                   // set color to red
    glBegin(GL_POLYGON);                        // draw a diamond
        glVertex2f(0.90, 0.50);
        glVertex2f(0.50, 0.90);
        glVertex2f(0.10, 0.50);
        glVertex2f(0.50, 0.10);
    glEnd();

    glColor3f(0.0, 0.0, 1.0);                   // set color to blue
    glRectf(0.25, 0.25, 0.75, 0.75);            // draw a rectangle

    glutSwapBuffers();                          // swap buffers
}
```

```
glBegin(mode);
    glVertex(v0); glVertex(v1); ...
glEnd();
```



Setting the Viewport in the Reshape Callback

```
void myReshape(int winWidth, int winHeight)       // reshape window
{
    ...
    glViewport (0, 0, winWidth, winHeight);       // reset the viewport
    ...
}
```

```
glClear(GL_COLOR_BUFFER_BIT);       // clear the window
glViewport (0, 0, w/2, h);          // set viewport to left half
  // ...drawing commands for the left half of window
glViewport (w/2, 0, w/2, h);        // set viewport to right half
  // ...drawing commands for the right half of window
glutSwapBuffers();                  // swap buffers
```

Setting a Two-Dimensional Projection

```
glMatrixMode(GL_PROJECTION);               // set projection matrix
glLoadIdentity();                          // initialize to identity
gluOrtho2D(0.0, 1.0, 0.0, 1.0);            // map unit square to viewport
```

glLoadIdentity(): Sets the current matrix to the identity matrix.

glLoadMatrix*(M): Loads (copies) a given matrix over the current matrix. (The '*' can be either 'f' or 'd' depending on whether the elements of M are GLfloat or GLdouble, respectively.)

glMultMatrix*(M): Post-multiplies the current matrix by a given matrix and replaces the current matrix with this result. Thus, if C is the current matrix on top of the stack, it will be replaced with the matrix product $C \cdot M$. (As above, the '*' can be either 'f' or 'd' depending on M.)

glPushMatrix(): Pushes a copy of the current matrix on top the stack. (Thus the stack now has two copies of the top matrix.)

glPopMatrix(): Pops the current matrix off the stack.

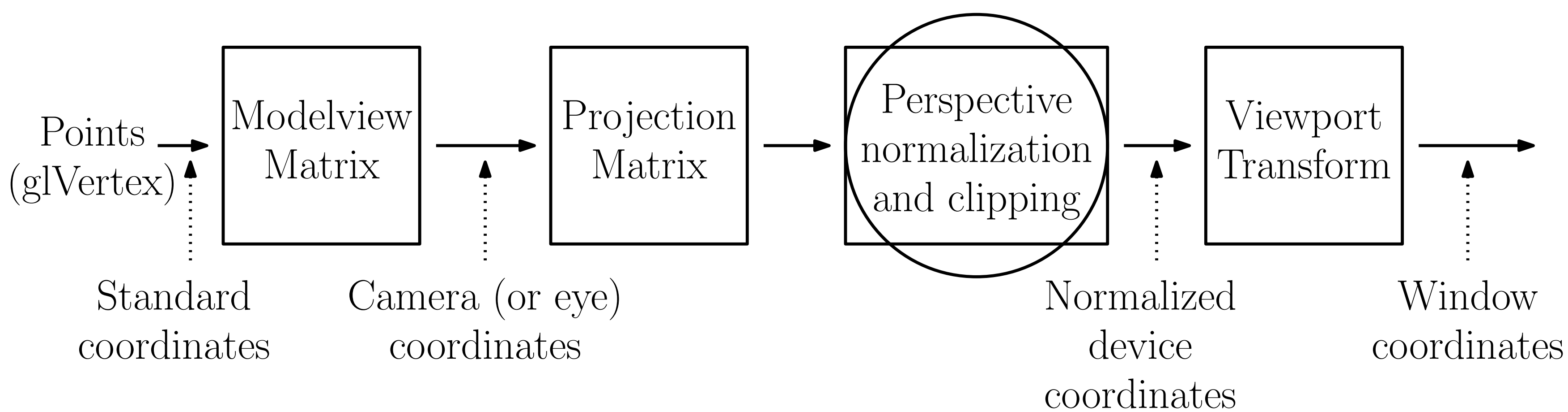


Fig. 23: Transformation pipeline.

(1) Push the matrix stack,

(2) Apply (i.e., multiply) all the desired transformation matrices with the current matrix, but *in the reverse order* from which you would like them to be applied to your object,

(3) Draw your object (the transformations will be applied automatically), and

(4) Pop the matrix stack.

Drawing an Rotated Rectangle (Correct)

```
glPushMatrix();                 // save the current matrix (M)
  glTranslatef(x, y, 0);        // apply translation (T)
  glRotatef(20, 0, 0, 1);       // apply rotation (R)
  glRectf(-2, -2, 2, 2);        // draw rectangle at the origin
glPopMatrix();                  // restore the old matrix (M)
```

Typical Structure of Redisplay Callback

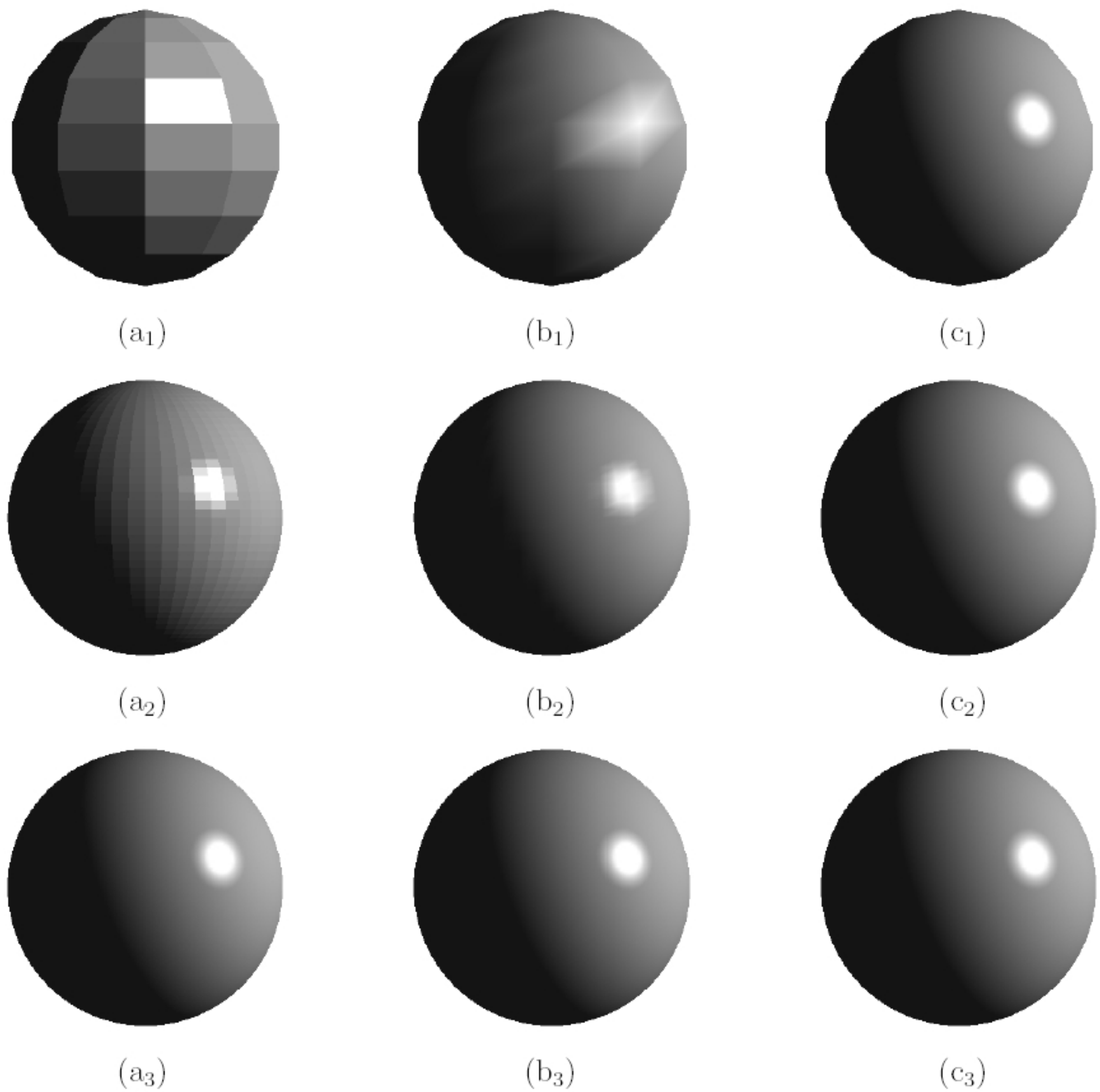
```
void myDisplay() {
                                         // clear the buffer
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();                      // start fresh
                                         // set up camera frame
  gluLookAt(eyeX, eyeY, eyeZ, atX, atY, atZ, upX, upY, upZ);
  myWorld.draw();                        // draw your scene
  glutSwapBuffers();                     // make it all appear
}
```

```
void myDisplay() {
  glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
  glLoadIdentity();
  gluLookAt( ... );                            // set up camera frame
  glMatrixMode(GL_PROJECTION);                 // set up projection
  glLoadIdentity();
  gluPerspective(fovy, aspect, near, far);    // or glFrustum(...)
  glMatrixMode(GL_MODELVIEW);
  myWorld.draw();                              // draw everything
  glutSwapBuffers();
}
```

```
glClearColor(0.0, 1.0, 0.0, 1.0);           // intentionally background
glEnable(GL_NORMALIZE);                     // normalize normal vectors
glShadeModel(GL_SMOOTH);                    // do smooth shading
glEnable(GL_LIGHTING);                      // enable lighting
                                            // ambient light (red)
GLfloat ambientIntensity[4] = {0.9, 0.0, 0.0, 1.0};
glLightModelfv(GL_LIGHT_MODEL_AMBIENT, ambientIntensity);

                                            // set up light 0 properties
GLfloat lt0Intensity[4] = {1.5, 1.5, 1.5, 1.0};  // white
glLightfv(GL_LIGHT0, GL_DIFFUSE, lt0Intensity);
glLightfv(GL_LIGHT0, GL_SPECULAR, lt0Intensity);

GLfloat lt0Position[4] = {2.0, 4.0, 5.0, 1.0};   // location
glLightfv(GL_LIGHT0, GL_POSITION, lt0Position);
                                           // attenuation params (a,b,c)
glLightf (GL_LIGHT0, GL_CONSTANT_ATTENUATION,  0.0);
glLightf (GL_LIGHT0, GL_LINEAR_ATTENUATION,    0.0);
glLightf (GL_LIGHT0, GL_QUADRATIC_ATTENUATION, 0.1);
glEnable(GL_LIGHT0);
```

Typical drawing with lighting

```
GLfloat color[]  = {0.0, 0.0, 1.0, 1.0};    // blue
GLfloat white[]  = {1.0, 1.0, 1.0, 1.0};    // white
                                            // set object colors
glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, color);
glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, white);
glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100);

glPushMatrix();
    glTranslatef(...);                      // your transformations
    glRotatef(...);
    glBegin(GL_POLYGON);                    // draw your shape
        glNormal3f(...); glVertex(...);     // remember to add normals
        glNormal3f(...); glVertex(...);
        glNormal3f(...); glVertex(...);
    glEnd();
glPopMatrix();
```



(a_1) (b_1) (c_1)
(a_2) (b_2) (c_2)
(a_3) (b_3) (c_3)

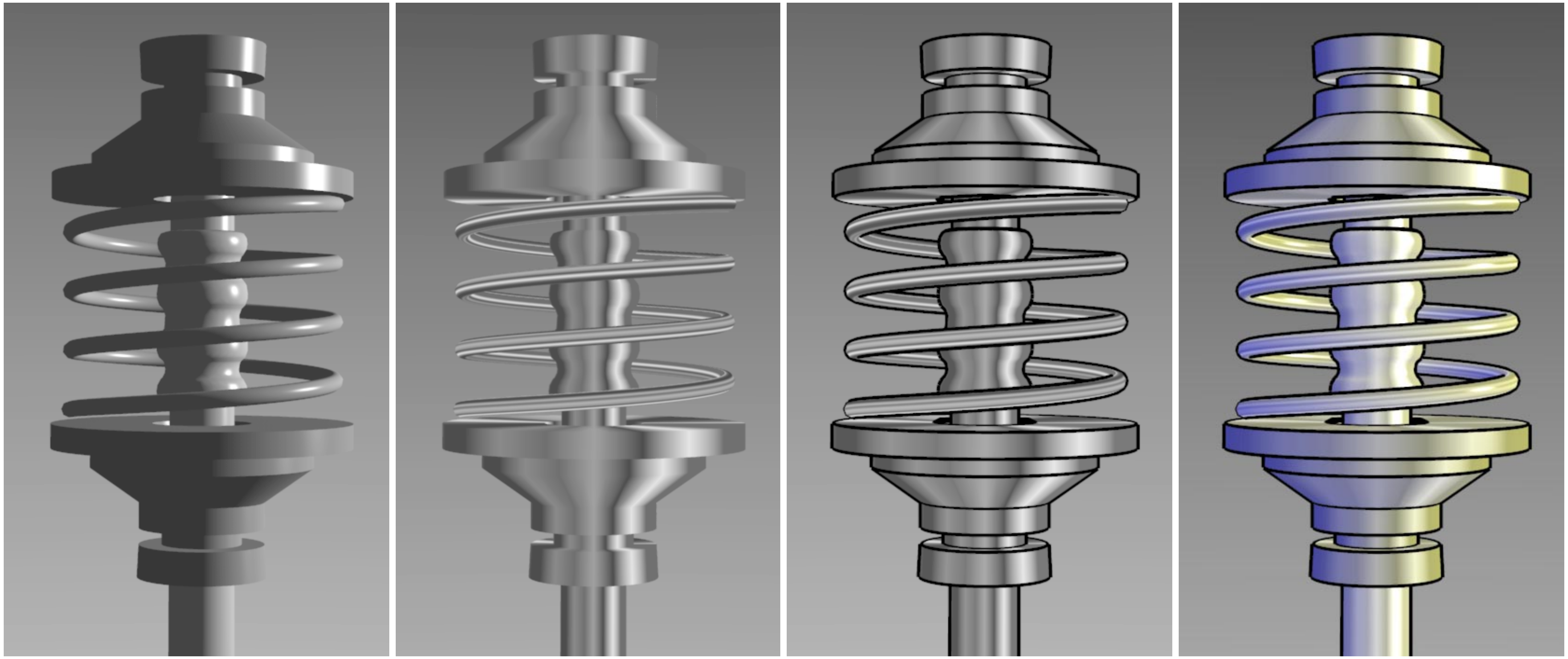


Figure 10: Left to Right: a) Phong shaded object. b) New metal-shaded object without edge lines. c) New metal-shaded object with edge lines. d) New metal-shaded object with a cool-to-warm shift.

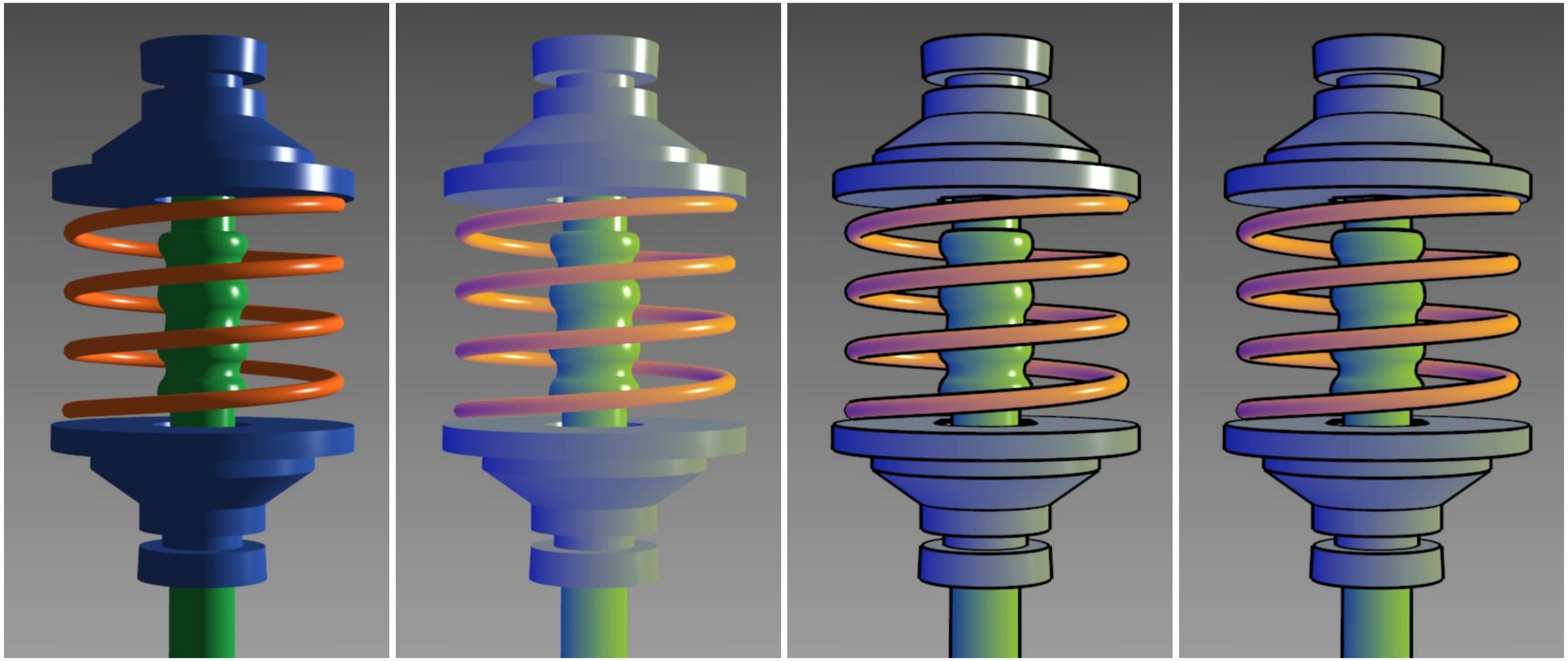


Figure 11: Left to Right: a) Phong model for colored object. b) New shading model with highlights, cool-to-warm hue shift, and without edge lines. c) New model using edge lines, highlights, and cool-to-warm hue shift. d) Approximation using conventional Phong shading, two colored lights, and edge lines.