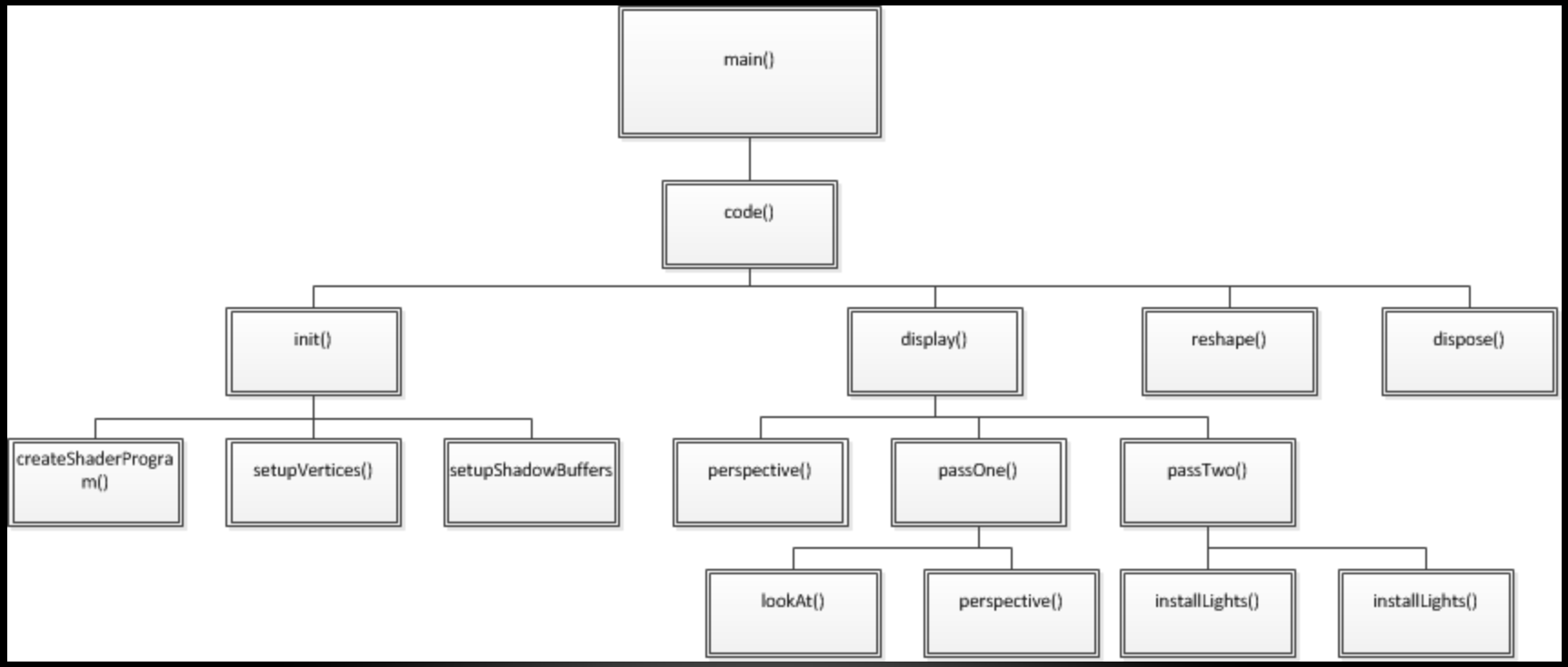
SHADOWS: THE CODE

OUTLINE


main()
code()
init()
display()
reshape()
dispose()
createShaderProgram()
setupVertices()
setupShadowBuffers
perspective()
passOne()
passTwo()
lookAt()
perspective()
installLights()
installLights()

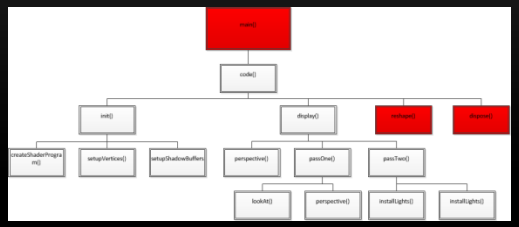
INSTANCE VARIABLES (1/2) – P. 1

```
private GLCanvas myCanvas;
private Material thisMaterial;
private String[] vBlinn1ShaderSource, vBlinn2ShaderSource, fBlinn2ShaderSource;
private int rendering_program1, rendering_program2;
private int vao[] = new int[1];
private int vbo[] = new int[4];
private int mv_location, proj_location, vertexLoc, n_location;
private float aspect;
private GLSLUtils util = new GLSLUtils();

// location of torus and camera
private Point3D torusLoc = new Point3D(1.6, 0.0, -0.3);
private Point3D pyrLoc = new Point3D(-1.0, 0.1, 0.3);
private Point3D cameraLoc = new Point3D(0.0, 0.2, 6.0);
private Point3D lightLoc = new Point3D(-3.8f, 2.2f, 1.1f);

private Matrix3D m_matrix = new Matrix3D();
private Matrix3D v_matrix = new Matrix3D();
private Matrix3D mv_matrix = new Matrix3D();
private Matrix3D proj_matrix = new Matrix3D();
```

INSTANCE VARIABLES (2/2) – PP 1-2

```
// light stuff
private float [] globalAmbient = new float[] { 0.7f, 0.7f, 0.7f, 1.0f };
private PositionalLight currentLight = new PositionalLight();

// shadow stuff
private int scSizeX, scSizeY;
private int [] shadow_tex = new int[1];
private int [] shadow_buffer = new int[1];
private Matrix3D lightV_matrix = new Matrix3D();
private Matrix3D lightP_matrix = new Matrix3D();
private Matrix3D shadowMVP1 = new Matrix3D();
private Matrix3D shadowMVP2 = new Matrix3D();
private Matrix3D b = new Matrix3D();

// model stuff
private ImportedModel pyramid = new ImportedModel("pyr.obj");
private Torus myTorus = new Torus(0.6f, 0.4f, 48);
private int numPyramidVertices, numTorusVertices;
```

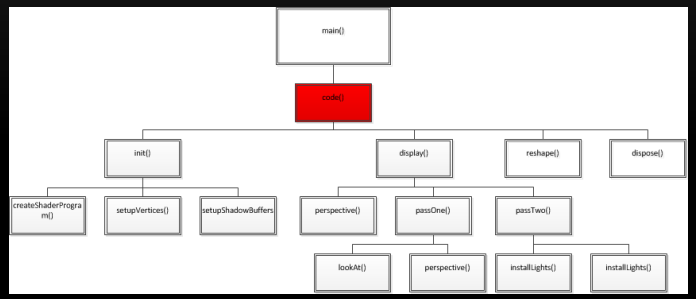
MAIN() METHOD – P. 9 – AS BEFORE DISPOSE() P. 9 AND RESHAPE() P. 7 HAVE CHANGED



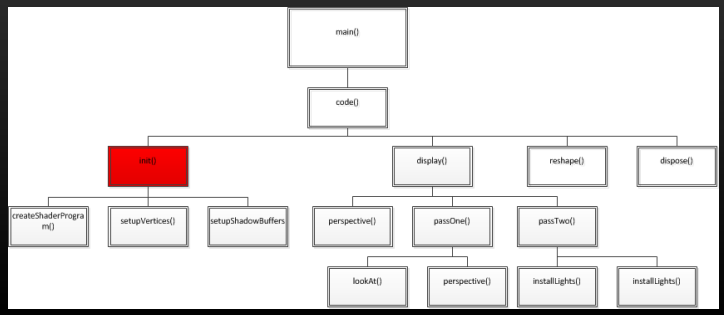
```
public static void main(String[] args)
{
    new Code();
}


public void dispose(GLAutoDrawable drawable)
{
    GL4 gl = (GL4) drawable.getGL();
    gl.glDeleteVertexArrays(1, vao, 0);
}


public void reshape(GLAutoDrawable drawable, int x, int y, int width, int height)
{
    GL4 gl = (GL4) GLContext.getCurrentGL();
    setupShadowBuffers();
}
```

CONSTRUCTOR FOR CODE() – P. 2

```
public Code()
{
    setTitle("Chapter8 - program 1");
    setSize(800, 800);
    myCanvas = new GLCanvas();
    myCanvas.addGLEventListener(this);
    getContentPane().add(myCanvas);
    setVisible(true);
    FPSAnimator animator = new FPSAnimator(myCanvas, 30);
    animator.start();
}
```

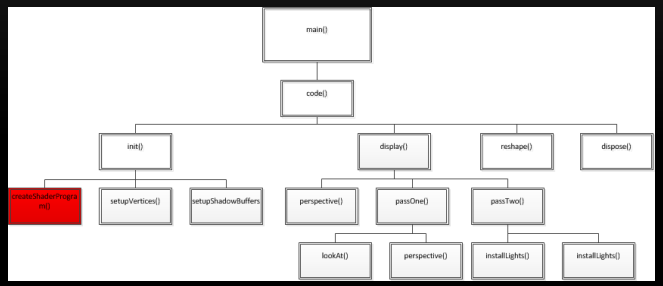


INIT() – P. 6




```
public void init(GLAutoDrawable drawable)
{
    GL4 gl = (GL4) GLContext.getCurrentGL();
    createShaderPrograms();
    setupVertices();
    setupShadowBuffers();

    b.setElementAt(0,0,0.5);b.setElementAt(0,1,0.0);b.setElementAt(0,2,0.0);b.setElementAt(0,3,0.5f);
    b.setElementAt(1,0,0.0);b.setElementAt(1,1,0.5);b.setElementAt(1,2,0.0);b.setElementAt(1,3,0.5f);
    b.setElementAt(2,0,0.0);b.setElementAt(2,1,0.0);b.setElementAt(2,2,0.5);b.setElementAt(2,3,0.5f);
    b.setElementAt(3,0,0.0);b.setElementAt(3,1,0.0);b.setElementAt(3,2,0.0);b.setElementAt(3,3,1.0f);

    // may reduce shadow border artifacts
    gl.glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_S, GL_CLAMP_TO_EDGE);
    gl.glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_WRAP_T, GL_CLAMP_TO_EDGE);
}
```

CREATESHADERPROGRAM() – PP. 9-10




```
private void createShaderPrograms()
{
    GL4 gl = (GL4) GLContext.getCurrentGL();
    int[] vertCompiled = new int[1];
    int[] fragCompiled = new int[1];

    vBlinn1ShaderSource = util.readShaderSource("code/blinnVert1.shader");
    vBlinn2ShaderSource = util.readShaderSource("code/blinnVert2.shader");
    fBlinn2ShaderSource = util.readShaderSource("code/blinnFrag2.shader");

    int vertexShader1 = gl.glCreateShader(GL_VERTEX_SHADER);
    int vertexShader2 = gl.glCreateShader(GL_VERTEX_SHADER);
    int fragmentShader2 = gl.glCreateShader(GL_FRAGMENT_SHADER);

    gl.glShaderSource(vertexShader1, vBlinn1ShaderSource.length, vBlinn1ShaderSource, null, 0);
    gl.glShaderSource(vertexShader2, vBlinn2ShaderSource.length, vBlinn2ShaderSource, null, 0);
    gl.glShaderSource(fragmentShader2, fBlinn2ShaderSource.length, fBlinn2ShaderSource, null, 0);

    gl.glCompileShader(vertexShader1);
    gl.glCompileShader(vertexShader2);
    gl.glCompileShader(fragmentShader2);

    rendering_program1 = gl.glCreateProgram();
    rendering_program2 = gl.glCreateProgram();

    gl.glAttachShader(rendering_program1, vertexShader1);
    gl.glAttachShader(rendering_program2, vertexShader2);
    gl.glAttachShader(rendering_program2, fragmentShader2);

    gl.glLinkProgram(rendering_program1);
    gl.glLinkProgram(rendering_program2);
}
```

VERTEX SHADER 1

```
#version 430

layout (location=0) in vec3 vertPos;

uniform mat4 shadowMVP;

void main(void)
{
    gl_Position = shadowMVP * vec4(vertPos,1.0);
}
```

VERTEX SHADER 2

```
#version 430

layout (location=0) in vec3 vertPos;
layout (location=1) in vec3 vertNormal;

out vec3 vNormal, vLightDir, vVertPos, vHalfVec;
out vec4 shadow_coord;

struct PositionalLight
{
    vec4 ambient, diffuse, specular;
    vec3 position;
};
 struct Material
{
    vec4 ambient, diffuse, specular;
    float shininess;
};

    uniform vec4 globalAmbient;
    uniform PositionalLight light;
    uniform Material material;
    uniform mat4 mv_matrix;
    uniform mat4 proj_matrix;
    uniform mat4 normalMat;
    uniform mat4 shadowMVP;
    layout (binding=0) uniform sampler2DShadow shadowTex;
```

VERTEX SHADER 2

```
void main(void)
{
    //output the vertex position to the rasterizer for interpolation
    vVertPos = (mv_matrix * vec4(vertPos,1.0)).xyz;

    //get a vector from the vertex to the light and output it to the rasterizer for interpolation
    vLightDir = light.position - vVertPos;

    //get a vertex normal vector in eye space and output it to the rasterizer for interpolation
    vNormal = (normalMat * vec4(vertNormal,1.0)).xyz;

    // calculate the half vector (L+V)
    vHalfVec = (vLightDir-vVertPos).xyz;

    shadow_coord = shadowMVP * vec4(vertPos,1.0);

    gl_Position = proj_matrix * mv_matrix * vec4(vertPos,1.0);
}
```

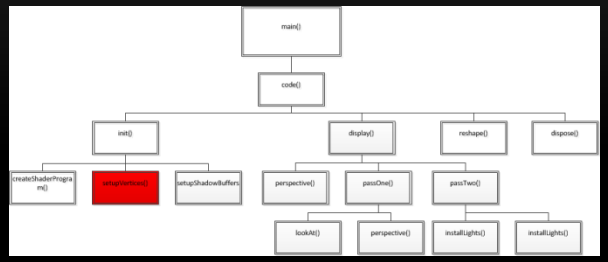
FRAGMENT SHADER 2

```
#version 430

in vec3 vNormal, vLightDir, vVertPos, vHalfVec;
in vec4 shadow_coord;
out vec4 fragColor;

struct PositionalLight
{
    vec4 ambient, diffuse, specular;
    vec3 position;
};

struct Material
{
    vec4 ambient, diffuse, specular;
    float shininess;
};

    uniform vec4 globalAmbient;
    uniform PositionalLight light;
    uniform Material material;
    uniform mat4 mv_matrix;
    uniform mat4 proj_matrix;
    uniform mat4 normalMat;
    uniform mat4 shadowMVP;
    layout (binding=0) uniform sampler2DShadow shadowTex;
```

FRAGMENT SHADER 2

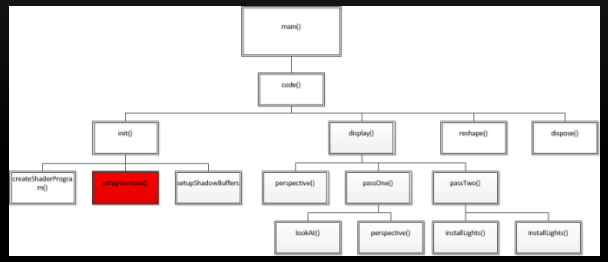
```
void main(void)
{
    vec3 L = normalize(vLightDir);
    vec3 N = normalize(vNormal);
    vec3 V = normalize(-vVertPos);
    vec3 H = normalize(vHalfVec);

    float inShadow = textureProj(shadowTex, shadow_coord);

    fragColor = globalAmbient * material.ambient + light.ambient * material.ambient;

    if (inShadow != 0.0)
    {
        fragColor += light.diffuse * material.diffuse * max(dot(L,N),0.0)
                   + light.specular * material.specular
                   * pow(max(dot(H,N),0.0),material.shininess*3.0);
    }
}
```

SETUPVERTICES() – (1/3), P. 7-8



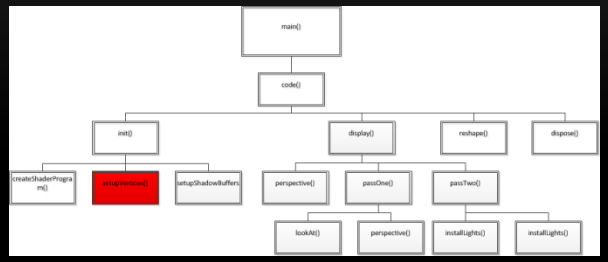

```
private void setupVertices()
{
    GL4 gl = (GL4) GLContext.getCurrentGL();

    // pyramid definition
    Vertex3D[] pyramid_vertices = pyramid.getVertices();
    numPyramidVertices = pyramid.getNumVertices();

    float[] pyramid_vertex_positions = new float[numPyramidVertices*3];
    float[] pyramid_normals = new float[numPyramidVertices*3];

    for (int i=0; i<numPyramidVertices; i++)
    {
        pyramid_vertex_positions[i*3]   = (float) (pyramid_vertices[i]).getX();
        pyramid_vertex_positions[i*3+1] = (float) (pyramid_vertices[i]).getY();
        pyramid_vertex_positions[i*3+2] = (float) (pyramid_vertices[i]).getZ();

        pyramid_normals[i*3]   = (float) (pyramid_vertices[i]).getNormalX();
        pyramid_normals[i*3+1] = (float) (pyramid_vertices[i]).getNormalY();
        pyramid_normals[i*3+2] = (float) (pyramid_vertices[i]).getNormalZ();
    }
```

SETUPVERTICES() – (2/3), P. 7-8



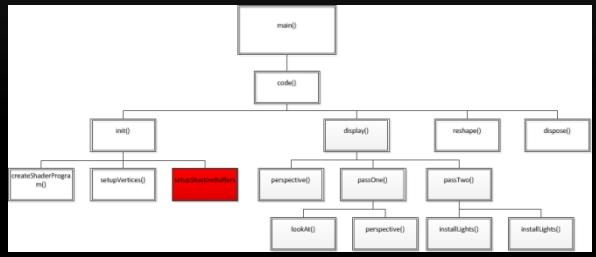

```
Vertex3D[] torus_vertices = myTorus.getVertices();

int[] torus_indices = myTorus.getIndices();
float[] torus_fvalues = new float[torus_indices.length*3];
float[] torus_nvalues = new float[torus_indices.length*3];

for (int i=0; i<torus_indices.length; i++)
{
    torus_fvalues[i*3]   = (float) (torus_vertices[torus_indices[i]]).getX();
    torus_fvalues[i*3+1] = (float) (torus_vertices[torus_indices[i]]).getY();
    torus_fvalues[i*3+2] = (float) (torus_vertices[torus_indices[i]]).getZ();

    torus_nvalues[i*3]   = (float) (torus_vertices[torus_indices[i]]).getNormalX();
    torus_nvalues[i*3+1] = (float) (torus_vertices[torus_indices[i]]).getNormalY();
    torus_nvalues[i*3+2] = (float) (torus_vertices[torus_indices[i]]).getNormalZ();
}
```

SETUPVERTICES – (2/3), P. 7-8



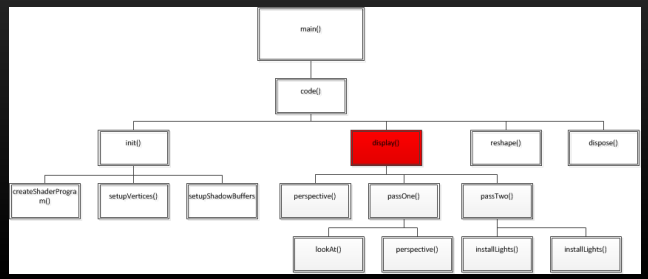

```
    numTorusVertices = torus_indices.length;

    gl.glGenVertexArrays(vao.length, vao, 0);
    gl.glBindVertexArray(vao[0]);

    gl.glGenBuffers(4, vbo, 0);

    //  put the Torus vertices into the first buffer,
    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
    FloatBuffer vertBuf = Buffers.newDirectFloatBuffer(torus_fvalues);
    gl.glBufferData(GL_ARRAY_BUFFER, vertBuf.limit()*4, vertBuf, GL_STATIC_DRAW);

    //  load the pyramid vertices into the second buffer
    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    FloatBuffer pyrVertBuf = Buffers.newDirectFloatBuffer(pyramid_vertex_positions);
    gl.glBufferData(GL_ARRAY_BUFFER, pyrVertBuf.limit()*4, pyrVertBuf, GL_STATIC_DRAW);

    // load the torus normal coordinates into the third buffer
    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
    FloatBuffer torusNorBuf = Buffers.newDirectFloatBuffer(torus_nvalues);
    gl.glBufferData(GL_ARRAY_BUFFER, torusNorBuf.limit()*4, torusNorBuf, GL_STATIC_DRAW);

    // load the pyramid normal coordinates into the fourth buffer
    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[3]);
    FloatBuffer pyrNorBuf = Buffers.newDirectFloatBuffer(pyramid_normals);
    gl.glBufferData(GL_ARRAY_BUFFER, pyrNorBuf.limit()*4, pyrNorBuf, GL_STATIC_DRAW);
}
```

SETUPSHADOWBUFFERS() – PP. 8-9




```
public void setupShadowBuffers()
{
    GL4 gl = (GL4) GLContext.getCurrentGL();
    scSizeX = myCanvas.getWidth();
    scSizeY = myCanvas.getHeight();

    gl.glGenFramebuffers(1, shadow_buffer, 0);

    gl.glGenTextures(1, shadow_tex, 0);
    gl.glBindTexture(GL_TEXTURE_2D, shadow_tex[0]);
    gl.glTexImage2D(GL_TEXTURE_2D, 0, GL_DEPTH_COMPONENT32,
    scSizeX, scSizeY, 0, GL_DEPTH_COMPONENT, GL_FLOAT, null);
    gl.glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_LINEAR);
    gl.glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_LINEAR);
    gl.glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_MODE, GL_COMPARE_REF_TO_TEXTURE);
    gl.glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_COMPARE_FUNC, GL_LEQUAL);
}
```

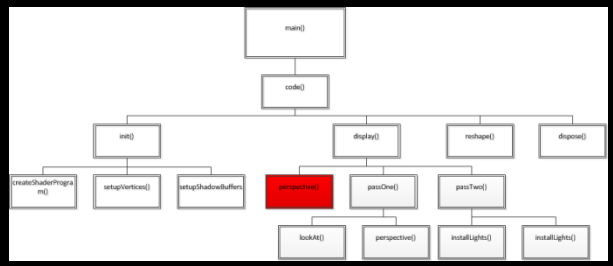
DISPLAY() – P. 2



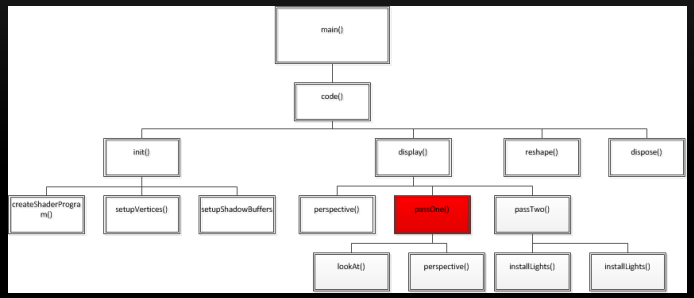

```
public void display(GLAutoDrawable drawable)
{
    GL4 gl = (GL4) GLContext.getCurrentGL();

    currentLight.setPosition(lightLoc);
    aspect = (float) myCanvas.getWidth() / (float) myCanvas.getHeight();
    proj_matrix = perspective(50.0f, aspect, 0.1f, 1000.0f);

    float bkg[] = { 0.0f, 0.0f, 0.0f, 1.0f };
    FloatBuffer bkgBuffer = Buffers.newDirectFloatBuffer(bkg);
    gl.glClearBufferfv(GL_COLOR, 0, bkgBuffer);

    gl.glBindFramebuffer(GL_FRAMEBUFFER, shadow_buffer[0]);
    gl.glFramebufferTexture(GL_FRAMEBUFFER, GL_DEPTH_ATTACHMENT, shadow_tex[0], 0);

    gl.glDrawBuffer(GL_NONE);
    gl.glEnable(GL_DEPTH_TEST);

    gl.glEnable(GL_POLYGON_OFFSET_FILL);// for reducing
    gl.glPolygonOffset(2.0f, 4.0f);//  shadow artifacts

    passOne();

    gl.glDisable(GL_POLYGON_OFFSET_FILL);// artifact reduction, continued

    gl.glBindFramebuffer(GL_FRAMEBUFFER, 0);
    gl.glActiveTexture(GL_TEXTURE0);
    gl.glBindTexture(GL_TEXTURE_2D, shadow_tex[0]);

    gl.glDrawBuffer(GL_FRONT);

    passTwo();
}
```

PERSPECTIVE() – P. 10

```
private Matrix3D perspective(float fovy, float aspect, float n, float f)
{
    float q = 1.0f / ((float) Math.tan(Math.toRadians(0.5f * fovy)));
    float A = q / aspect;
    float B = (n + f) / (n - f);
    float C = (2.0f * n * f) / (n - f);
    Matrix3D r = new Matrix3D();
    r.setElementAt(0,0,A);
    r.setElementAt(1,1,q);
    r.setElementAt(2,2,B);
    r.setElementAt(3,2,-1.0f);
    r.setElementAt(2,3,C);
    r.setElementAt(3,3,0.0f);
    return r;
}
```

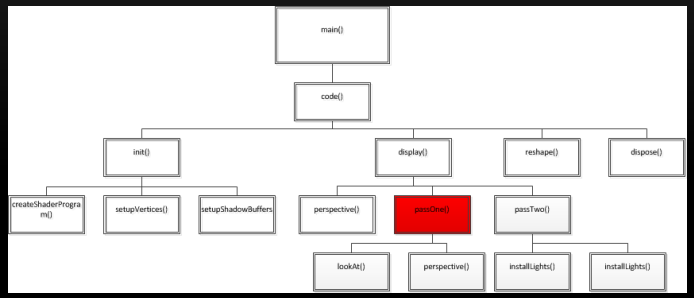


PASSONE() – PP. 2-4, (1/3)




```
public void passOne()
{
    GL4 gl = (GL4) GLContext.getCurrentGL();

    gl.glUseProgram(rendering_program1);

    Point3D origin = new Point3D(0.0, 0.0, 0.0);
    Vector3D up = new Vector3D(0.0, 1.0, 0.0);
    lightV_matrix.setToIdentity();
    lightP_matrix.setToIdentity();

    lightV_matrix = lookAt(currentLight.getPosition(), origin, up);// vector from light to origin
    lightP_matrix = perspective(50.0f, aspect, 0.1f, 1000.0f);

    // draw the torus
    m_matrix.setToIdentity();
    m_matrix.translate(torusLoc.getX(),torusLoc.getY(),torusLoc.getZ());
    m_matrix.rotateX(25.0);

    shadowMVP1.setToIdentity();
    shadowMVP1.concatenate(lightP_matrix);
    shadowMVP1.concatenate(lightV_matrix);
    shadowMVP1.concatenate(m_matrix);
    int shadow_location = gl.glGetUniformLocation(rendering_program1, "shadowMVP");
    gl.glUniformMatrix4fv(shadow_location, 1, false, shadowMVP1.getFloatValues(), 0);
```

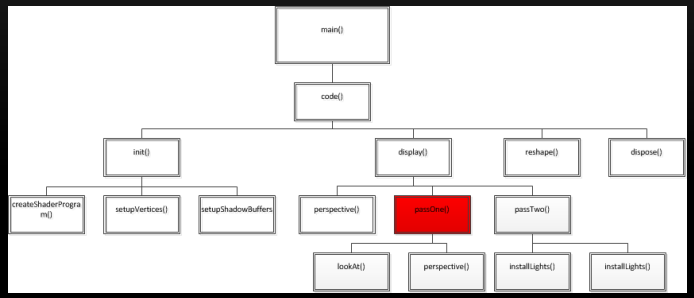
PASSONE() – PP. 2-4, (2/3)



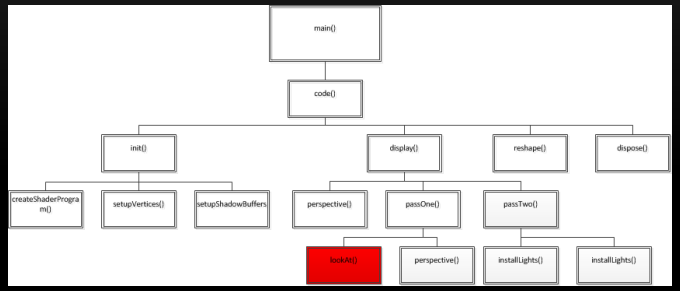

```
// set up torus vertices buffer
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
gl.glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
gl.glEnableVertexAttribArray(0);

gl.glClear(GL_DEPTH_BUFFER_BIT);
gl.glEnable(GL_CULL_FACE);
gl.glFrontFace(GL_CCW);
gl.glEnable(GL_DEPTH_TEST);
gl.glDepthFunc(GL_LEQUAL);

gl.glDrawArrays(GL_TRIANGLES, 0, numTorusVertices);

// ---- draw the pyramid
//  build the MODEL matrix
m_matrix.setToIdentity();
m_matrix.translate(pyrLoc.getX(),pyrLoc.getY(),pyrLoc.getZ());
m_matrix.rotateX(30.0);
m_matrix.rotateY(40.0);

shadowMVP1.setToIdentity();
shadowMVP1.concatenate(lightP_matrix);
shadowMVP1.concatenate(lightV_matrix);
shadowMVP1.concatenate(m_matrix);

gl.glUniformMatrix4fv(shadow_location, 1, false, shadowMVP1.getFloatValues(), 0);
```

PASSONE() – PP. 2-4, (3/3)

```
    // set up vertices buffer
    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    gl.glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
    gl.glEnableVertexAttribArray(0);

    gl.glEnable(GL_CULL_FACE);
    gl.glFrontFace(GL_CCW);
    gl.glEnable(GL_DEPTH_TEST);
    gl.glDepthFunc(GL_LEQUAL);

    gl.glDrawArrays(GL_TRIANGLES, 0, pyramid.getNumVertices());
}
```



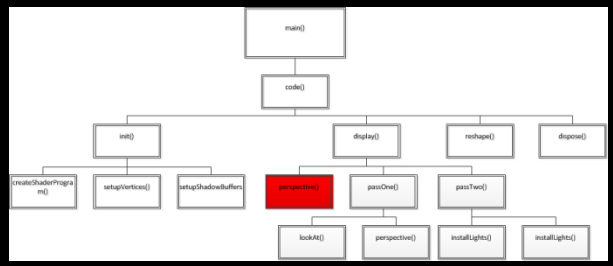
LOOKAT() – P.P. 10-11



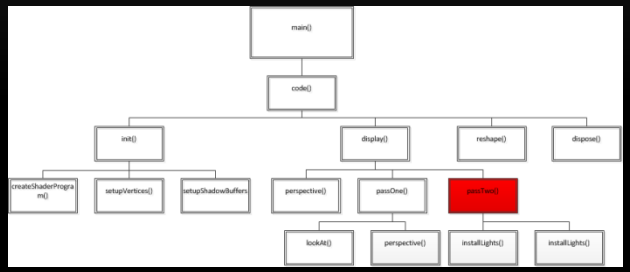

```
private Matrix3D lookAt(Point3D eye, Point3D target, Vector3D y)
{
    Vector3D eyeV = new Vector3D(eye);
    Vector3D targetV = new Vector3D(target);
    Vector3D fwd = (targetV.minus(eyeV)).normalize();
    Vector3D side = (fwd.cross(y)).normalize();
    Vector3D up = (side.cross(fwd)).normalize();
    Matrix3D look = new Matrix3D();
    look.setElementAt(0,0, side.getX());
    look.setElementAt(1,0, up.getX());
    look.setElementAt(2,0, -fwd.getX());
    look.setElementAt(3,0, 0.0f);
    look.setElementAt(0,1, side.getY());
    look.setElementAt(1,1, up.getY());
    look.setElementAt(2,1, -fwd.getY());
    look.setElementAt(3,1, 0.0f);
    look.setElementAt(0,2, side.getZ());
    look.setElementAt(1,2, up.getZ());
    look.setElementAt(2,2, -fwd.getZ());
    look.setElementAt(3,2, 0.0f);
    look.setElementAt(0,3, side.dot(eyeV.mult(-1)));
    look.setElementAt(1,3, up.dot(eyeV.mult(-1)));
    look.setElementAt(2,3, (fwd.mult(-1)).dot(eyeV.mult(-1)));
    look.setElementAt(3,3, 1.0f);
    return(look);
}
```

PASSONE ALSO CALLS PERSPECTIVE AGAIN

```
private Matrix3D perspective(float fovy, float aspect, float n, float f)
{
    float q = 1.0f / ((float) Math.tan(Math.toRadians(0.5f * fovy)));
    float A = q / aspect;
    float B = (n + f) / (n - f);
    float C = (2.0f * n * f) / (n - f);
    Matrix3D r = new Matrix3D();
    r.setElementAt(0,0,A);
    r.setElementAt(1,1,q);
    r.setElementAt(2,2,B);
    r.setElementAt(3,2,-1.0f);
    r.setElementAt(2,3,C);
    r.setElementAt(3,3,0.0f);
    return r;
}
```

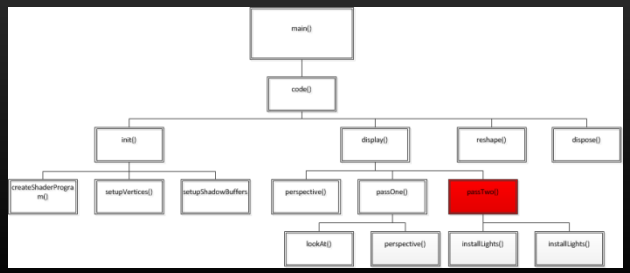


PASSTWO() – PP. 4-6 (1/6)



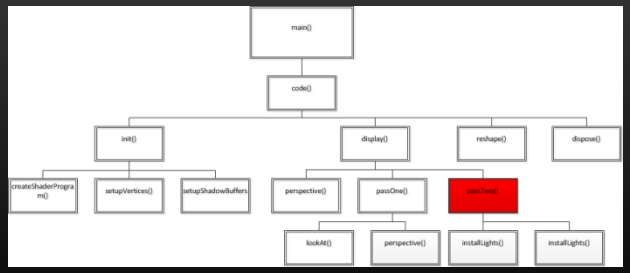

```
public void passTwo()
{
    GL4 gl = (GL4) GLContext.getCurrentGL();

    gl.glUseProgram(rendering_program2);

    // draw the torus
    thisMaterial = graphicslib3D.Material.BRONZE;

    mv_location = gl.glGetUniformLocation(rendering_program2, "mv_matrix");
    proj_location = gl.glGetUniformLocation(rendering_program2, "proj_matrix");
    n_location = gl.glGetUniformLocation(rendering_program2, "normalMat");
    int shadow_location = gl.glGetUniformLocation(rendering_program2,  "shadowMVP");

    //  build the MODEL matrix
    m_matrix.setToIdentity();
    m_matrix.translate(torusLoc.getX(),torusLoc.getY(),torusLoc.getZ());
    m_matrix.rotateX(25.0);
```

PASSTWO() – PP. 4-6 (2/6)




```
//  build the VIEW matrix
v_matrix.setToIdentity();
v_matrix.translate(-cameraLoc.getX(),-cameraLoc.getY(),-cameraLoc.getZ());

installLights(rendering_program2, v_matrix);

//  build the MODEL-VIEW matrix
mv_matrix.setToIdentity();
mv_matrix.concatenate(v_matrix);
mv_matrix.concatenate(m_matrix);

shadowMVP2.setToIdentity();
shadowMVP2.concatenate(b);
shadowMVP2.concatenate(lightP_matrix);
shadowMVP2.concatenate(lightV_matrix);
shadowMVP2.concatenate(m_matrix);
```

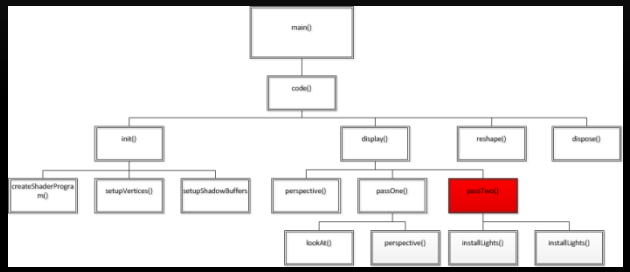
PASSTWO() – PP. 4-6 (3/6)




```
//  put the MV and PROJ matrices into the corresponding uniforms
gl.glUniformMatrix4fv(mv_location, 1, false, mv_matrix.getFloatValues(), 0);
gl.glUniformMatrix4fv(proj_location, 1, false, proj_matrix.getFloatValues(), 0);
gl.glUniformMatrix4fv(n_location, 1, false, (mv_matrix.inverse()).transpose().getFloatValues(),
                      0);
gl.glUniformMatrix4fv(shadow_location, 1, false, shadowMVP2.getFloatValues(), 0);

// set up torus vertices buffer
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[0]);
gl.glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
gl.glEnableVertexAttribArray(0);
```

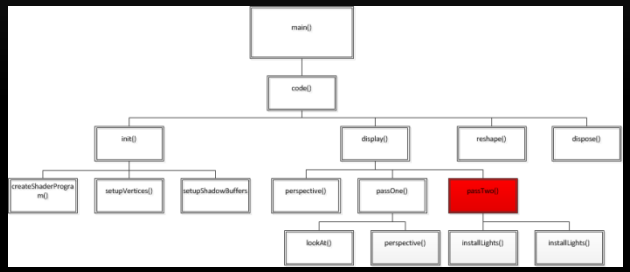
PASSTWO() – PP. 4-6 (4/6)

```
// set up torus normals buffer
gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[2]);
gl.glVertexAttribPointer(1, 3, GL_FLOAT, false, 0, 0);
gl.glEnableVertexAttribArray(1);

gl.glClear(GL_DEPTH_BUFFER_BIT);
gl.glEnable(GL_CULL_FACE);
gl.glFrontFace(GL_CCW);
gl.glEnable(GL_DEPTH_TEST);
gl.glDepthFunc(GL_LEQUAL);

gl.glDrawArrays(GL_TRIANGLES, 0, numTorusVertices);
```

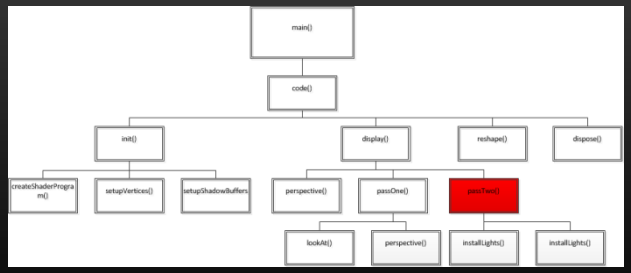


PASSTWO() – PP. 4-6 (5/6)

```
// draw the pyramid
thisMaterial = graphicslib3D.Material.GOLD;
installLights(rendering_program2, v_matrix);

//  build the MODEL matrix
m_matrix.setToIdentity();
m_matrix.translate(pyrLoc.getX(),pyrLoc.getY(),pyrLoc.getZ());
m_matrix.rotateX(30.0);
m_matrix.rotateY(40.0);

//  build the MODEL-VIEW matrix
mv_matrix.setToIdentity();
mv_matrix.concatenate(v_matrix);
mv_matrix.concatenate(m_matrix);

shadowMVP2.setToIdentity();
shadowMVP2.concatenate(b);
shadowMVP2.concatenate(lightP_matrix);
shadowMVP2.concatenate(lightV_matrix);
shadowMVP2.concatenate(m_matrix);
gl.glUniformMatrix4fv(shadow_location, 1, false, shadowMVP2.getFloatValues(), 0);
```



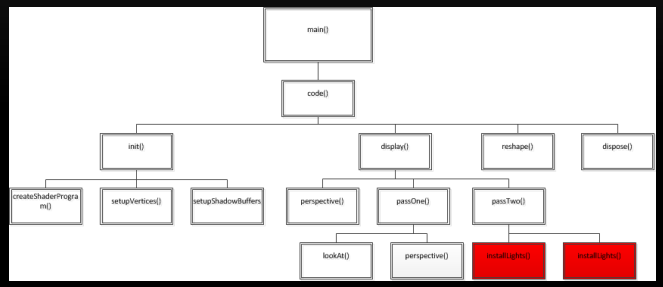
PASSTWO() – PP. 4-6 (6/6)



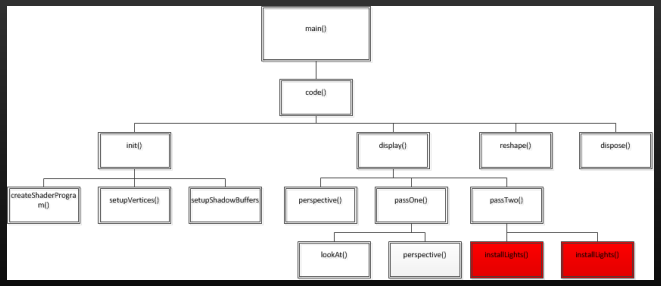

```
    //  put the MV and PROJ matrices into the corresponding uniforms
    gl.glUniformMatrix4fv(mv_location, 1, false, mv_matrix.getFloatValues(), 0);
    gl.glUniformMatrix4fv(proj_location, 1, false, proj_matrix.getFloatValues(), 0);
    gl.glUniformMatrix4fv(n_location, 1, false, (mv_matrix.inverse()).transpose().getFloatValues(), 0);

    // set up vertices buffer
    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[1]);
    gl.glVertexAttribPointer(0, 3, GL_FLOAT, false, 0, 0);
    gl.glEnableVertexAttribArray(0);

    // set up normals buffer
    gl.glBindBuffer(GL_ARRAY_BUFFER, vbo[3]);
    gl.glVertexAttribPointer(1, 3, GL_FLOAT, false, 0, 0);
    gl.glEnableVertexAttribArray(1);

    gl.glEnable(GL_CULL_FACE);
    gl.glFrontFace(GL_CCW);
    gl.glEnable(GL_DEPTH_TEST);
    gl.glDepthFunc(GL_LEQUAL);

    gl.glDrawArrays(GL_TRIANGLES, 0, pyramid.getNumVertices());
}
```

INSTALLLIGHTS() – PP. 8-9 (1/3)

```
private void installLights(int rendering_program, Matrix3D v_matrix)
{
    GL4 gl = (GL4) GLContext.getCurrentGL();

    Material currentMaterial = new Material();
    currentMaterial = thisMaterial;

    Point3D lightP = currentLight.getPosition();
    Point3D lightPv = lightP.mult(v_matrix);

    float [] currLightPos = new float[] { (float) lightPv.getX(),
    (float) lightPv.getY(),
    (float) lightPv.getZ() };

    // get the location of the global ambient light field in the shader
    int globalAmbLoc = gl.glGetUniformLocation(rendering_program, "globalAmbient");

    // set the current globalAmbient settings
    gl.glProgramUniform4fv(rendering_program, globalAmbLoc, 1, globalAmbient, 0);
```

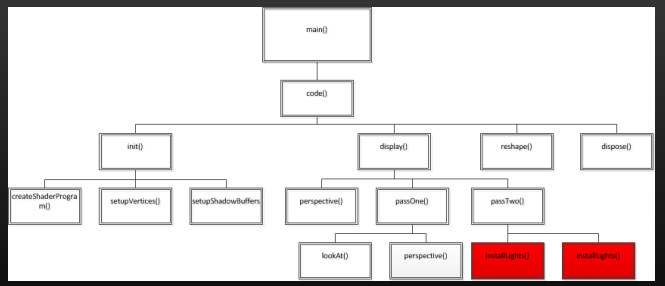


INSTALLLIGHTS() – PP. 8-9 (2/3)



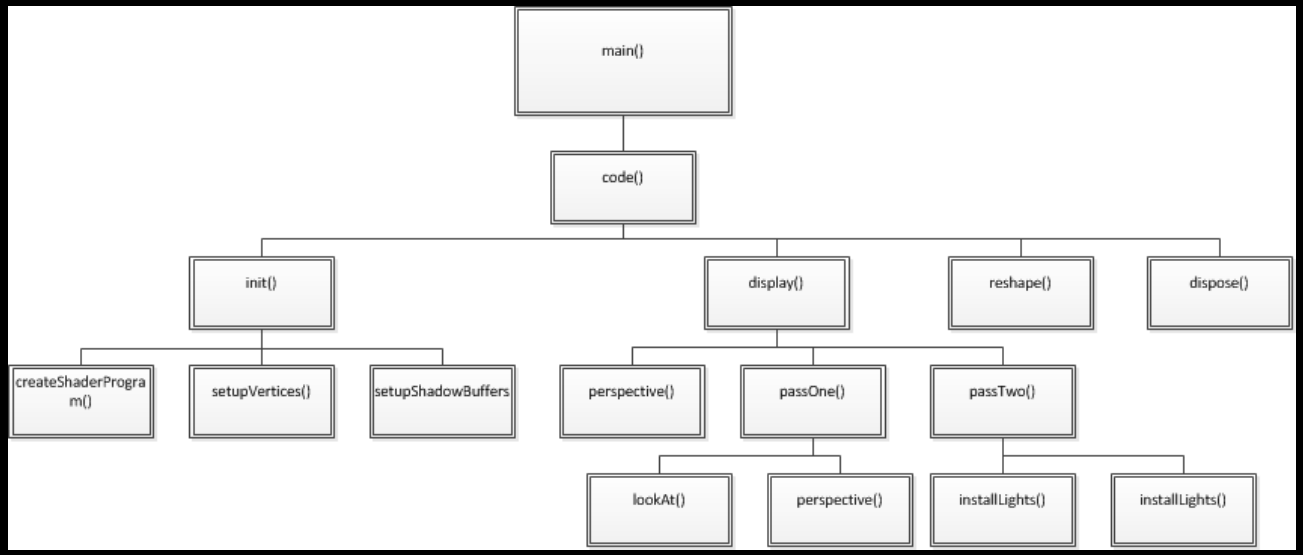

```
// get the locations of the light and material fields in the shader
int ambLoc = gl.glGetUniformLocation(rendering_program, "light.ambient");
int diffLoc = gl.glGetUniformLocation(rendering_program, "light.diffuse");
int specLoc = gl.glGetUniformLocation(rendering_program, "light.specular");
int posLoc = gl.glGetUniformLocation(rendering_program, "light.position");

int MambLoc = gl.glGetUniformLocation(rendering_program, "material.ambient");
int MdiffLoc = gl.glGetUniformLocation(rendering_program, "material.diffuse");
int MspecLoc = gl.glGetUniformLocation(rendering_program, "material.specular");
int MshiLoc = gl.glGetUniformLocation(rendering_program, "material.shininess");
```

INSTALLLIGHTS() – PP. 8-9 (3/3)




```
    // set the uniform light and material values in the shader
    gl.glProgramUniform4fv(rendering_program, ambLoc, 1, currentLight.getAmbient(), 0);
    gl.glProgramUniform4fv(rendering_program, diffLoc, 1, currentLight.getDiffuse(), 0);
    gl.glProgramUniform4fv(rendering_program, specLoc, 1, currentLight.getSpecular(), 0);
    gl.glProgramUniform3fv(rendering_program, posLoc, 1, currLightPos, 0);

    gl.glProgramUniform4fv(rendering_program, MambLoc, 1, currentMaterial.getAmbient(), 0);
    gl.glProgramUniform4fv(rendering_program, MdiffLoc, 1, currentMaterial.getDiffuse(), 0);
    gl.glProgramUniform4fv(rendering_program, MspecLoc, 1, currentMaterial.getSpecular(), 0);
    gl.glProgramUniform1f(rendering_program, MshiLoc, currentMaterial.getShininess());
}
```

SUMMARY


main()
code()
init()
display()
reshape()
dispose()
createShaderProgram()
setupVertices()
setupShadowBuffers
perspective()
passOne()
passTwo()
lookAt()
perspective()
installLights()
installLights()