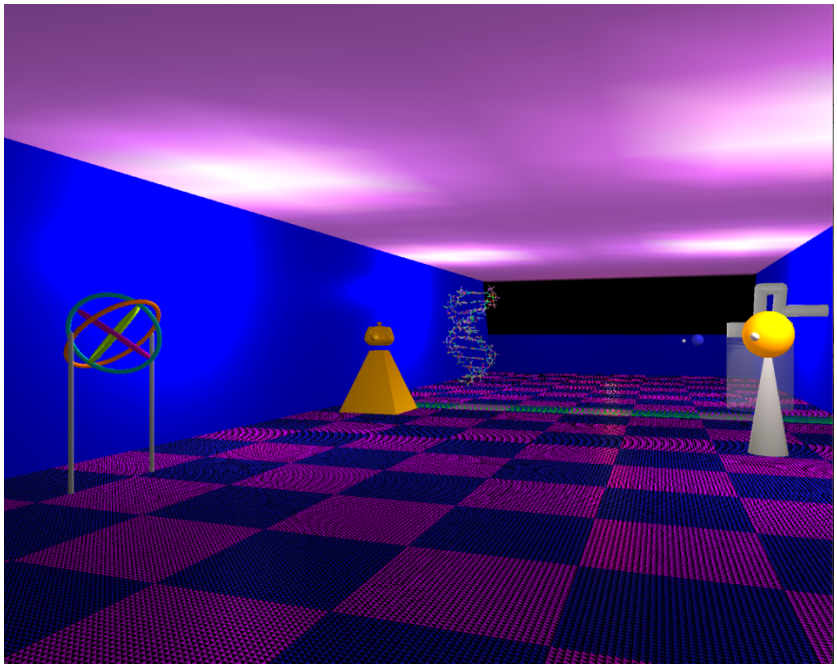
Bora Soydan 76741, Ata Can Taşdan 76216

COMP 410 Project Report: Interactive 3D Virtual Walkthrough Of A Museum

Teammates: Bora Soydan 76741, Ata Can Taşdan 76216



Introduction

We have designed and implemented an intreactive 3D virtual walkthrough of museum This museum is a first-person gallery application written entirely in C with modern OpenGL and GLUT frameworks. Inside this constructed room and surrounding landscape, users can move freely with various functionalities such as walking, jumping, zooming in, zooming out and so on. There are eight different lights, textured surfaces that can be adjusted by the user which creates a immersive visual atmosphere. In the museum, there are five different sculptures, eah are implemented by timer-based parametric equations that highlight distinct computer-graphics concepts such as translation, rotation, projection, lighting and texture mapping. Each of these graphics concepts can be arranged by the users like and interest. Besides visual features, the museum includes auditory features such as pouring sound of the tea a and music while walking in the museum. The user can add new objects, new functionalities etc. It is all up to the users creativity and interactivity.

Motivation

We have picked this project because we thought that it can give a clear, code-level look at how an interactive 3-D program really works. Instead of using Unity or Unreal, every feature (camera movement, lighting, textures, and animation) is written directly in modern OpenGL and C. By doing so, the museum makes the graphics pipeline easy to follow: load a texture, set a material, build a transformation matrix, draw the object and repeat. One of the most important reasons of choosing this project is the techniques used in the sculptures such as rotation, lighting etc. and the functionalities, animations used in the museum contain siginificant amount of knowledge of COMP 410. Moreover, building a museum offers a great opportunity for the user to show their creativity.

Tools Used And Why

The museum is built entirely with C and modern OpenGL to keep full, low-level control over the rendering pipeline. They provide the knowledge of writing the vertex transformations, material settings and draw calls which directly makes every graphics concept visible in code. GLUT supplies the bare-bones window, input and timer callbacks (lightweight, cross-platform and perfect for quick prototypes) while GLU (specifically its quadric helpers) generates spheres, cylinders, and disks without hand-rolling tessellation maths. Textures are loaded through a custom pngLoader wrapper around libpng, giving lossless images and mipmap generation without pulling in a heavyweight asset library. For sound, the program simply forks macOS's afplay command, an easy way to stream WAV files asynchronously without adding an audio SDK. Together, this toolchain stays small, portable, and transparent. Every subsystem can be read in a few source files, recompiled in seconds and modified line by line for teaching or experimentation. Also, the knowledge we gained for the OpenGL framework through the lessons and homeworks made us choose the OpenGL framework for implementing the project with confidence

Implementation Order

For the project, we have followed an order to implement it. Implementing the project layer by layer gave us the opportunity for better testing, debugging and it allowed us to build the project on top of each other. The order of our implementation is as follows:

1) We first created a minimal GLUT window and placed a fixed-function camera so we could render a single colored triangle. This confirmed context creation and input callbacks.

2) We laid down the room shell (floor, walls, ceiling) using tiled quads so we had visible geometry to walk around.

3) We enabled OpenGL's built-in lighting engine and added eight lights, tuning their attenuation before wiring up per-light on/off keys.

4) A tiny pngLoader wrapper went in next, letting us texture the ceiling and far wall. Once textures worked, we generalised the loader so other assets could be dropped in later.

5) With the environment finished, we implemented each sculpture as its own function pair (drawSculptureX and updateSculptureX), testing them one at a time.

6) After the sculptures animated correctly, we did the navigation part, giving keyboard movement and mouse look.

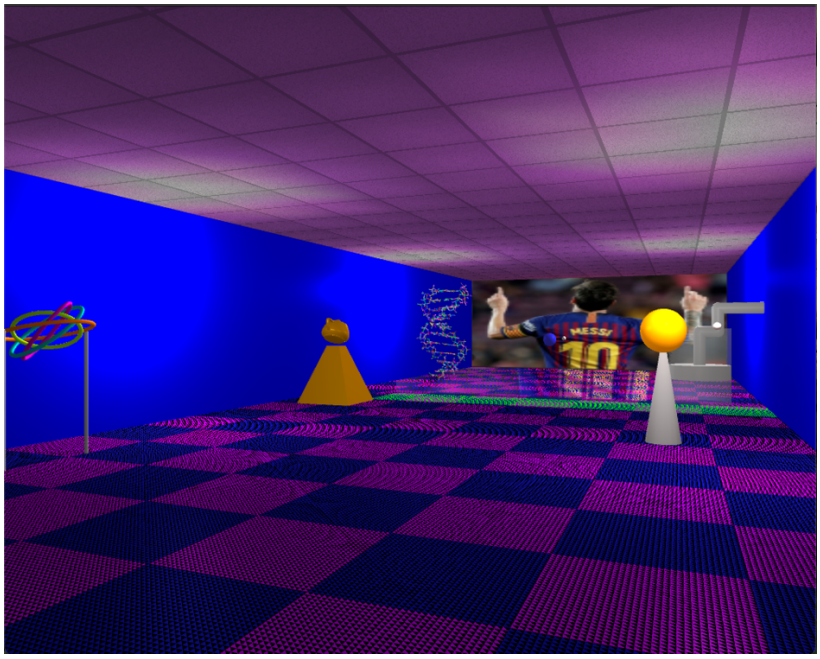
7) Lastly, when the basis of the project was done, we tried to use our creativity and tried to improve the museum with auditory effects and new added motions to objects.

Navigation (How visitors move through the museum)

The museum uses a small, self-contained navigation engine (navigator.c) that creates the GLUT window, sets a perspective frustum and maintains a single movable camera. Position (cameraLocX/Y/Z), yaw–pitch angles (rotationH, rotationV), and zoom level live in globals so every subsystem can read them. Keyboard and mouse events do nothing more than flip Boolean “smooth-motion” flags. A timer function (navSmoothMotion) polls those flags every 75 ms, affecting the camera by fixed increments. Arrow keys strafe or turn, j and d launch jump/duck arcs and + / – call navZoom(). The current view matrix is rebuilt every redraw in navUpdateCamera() and the rest of the program simply draws the scene from that viewpoint.

Design Of The Museum

The room is a parametrised box built entirely at run-time. We treat width, length, and height (constants in scimus.h) as a virtual grid and fill each surface with quads whose size is “TILE_RES” (64 or 512 world units). Every face is drawn in its own helper (drawFloor, drawCeiling, drawWalls) so the geometry, normals, materials, and textures can be tuned independently. Because each helper runs inside a “glPushMatrix … glPopMatrix” pair, we can translate and rotate the local coordinate frame once and then emit vertices with simple loop math. This keeps the world calculations minimal and guarantees all tiles line up perfectly at the edges.



- **Floor:** "glTranslated(–ROOM_WIDTH/2, –FLOOR_LEVEL, –ROOM_LENGTH/2)" drops the grid into place after "glRotated(180,1,0,0)" flips it so that the +y normals point into the room.

For the tiling, two nested loops iterate:

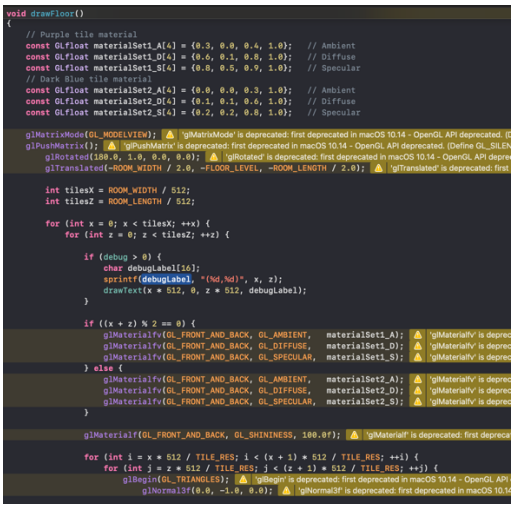
for x = 0. ROOM_WIDTH / 512
 for z = 0 … ROOM_LENGTH / 512

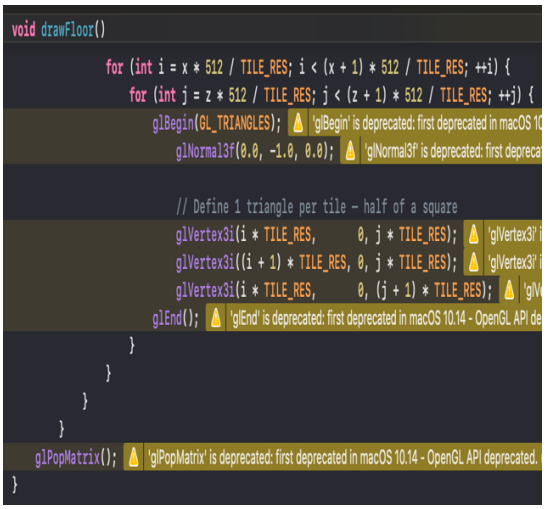
Each pass splits the 512x512 square into smaller "TILE_RES" sub-triangles using a second set of loops, and these are subsequently output as "GL_TRIANGLES."

The materials tiles alternate between sets of dark-blue and purple materials using "(x+z) % 2," forming a checkerboard that helps visitors understand scale.

In order for the lighting engine to evenly shade the floor, each vertex uses (0,-1,0) for the normals (pointing down in the pre-flipped frame).

DrawText() prints the (x,z) indices of each tile on the surface for debugging if "If debug>0."
-"glRotated(180,1,0,0)" is the floor.



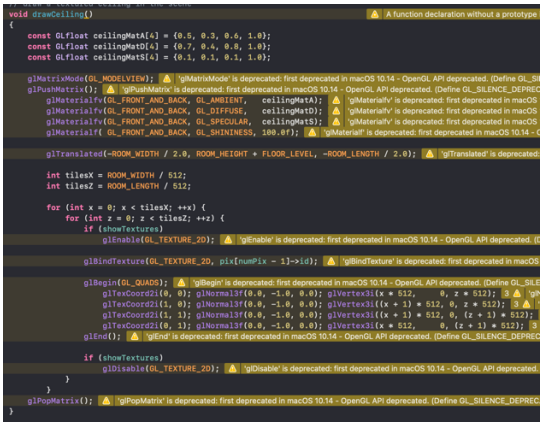


- **Ceiling:** Since the visible face already points downward, all that needs to be done for transformation is to use “glTranslated(-ROOM_WIDTH/2, ROOM_HEIGHT+FLOOR_LEVEL, -ROOM_LENGTH/2) (no rotation)”.

  Tiles are rendered as single “GL_QUADS”, but the geometry has the same outer loops as the floor.

  A texture option is available. The code assigns “UVs (0,0)…(1,1)”, binds “pix[numPix-1]->id (ceiling PNG)”, and activates “GL_TEXTURE_2D” when “showTextures” is “true”, ensuring that each 512×512 tile displays a single copy of the image. When the flag is disabled, the material turns back to mauve.

  Once more, the normals are (0,-1,0). They are actually pointing in the direction of the room now.



- **Walls:** The walls reuse a helper pattern but in different ways.

```
void drawWalls()
{
    int i, j;

    // material properties
    GLfloat const colorA[4] = {0.0, 0.0, 0.4, 1.0}; // Ambient
    GLfloat const colorD[4] = {0.0, 0.0, 0.6, 1.0}; // Diffuse
    GLfloat const colorS[4] = {0.0, 0.0, 0.8, 1.0}; // Specular

    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,   colorA);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,   colorD);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  colorS);
    glMaterialf( GL_FRONT_AND_BACK, GL_SHININESS, 100.0f);
```

Right & Left: They rotate ±90° about y to face +z and translate to lower-left corner. The outer loop over length columns “(ROOM_LENGTH/TILE_RES)”, inner loop over height rows “(ROOM_HEIGHT/TILE_RES)”, drawn as “GL_QUAD_STRIP”. There are no textures, only royal-blue material.

Near Wall: It rotates 180° about y and translate to near “z=+ROOM_LENGTH/2”. It uses the same “GL_QUAD_STRIP” grid. It has the plain material.

```
void drawWalls()
    // draw near wall
    glMatrixMode(GL_MODELVIEW);  'glMatrixMode' is deprecated: first depr
    glPushMatrix();  'glPushMatrix' is deprecated: first deprecated in macO

    // rotate to face wall
    glRotated(180.0, 0.0, 1.0, 0.0);  'glRotated' is deprecated: first d

    // move to lower left corner
    glTranslated(ROOM_WIDTH/-2.0, FLOOR_LEVEL, ROOM_LENGTH/-2.0);

    // draw wall panels
    for (i = 0; i < ROOM_WIDTH/TILE_RES; ++i) {
        glBegin(GL_QUAD_STRIP);  'glBegin' is deprecated: first deprecate
        for (j = 0; j <= ROOM_HEIGHT/TILE_RES; ++j) {
            glNormal3f(0.0, 0.0, 1.0);  'glNormal3f' is deprecated: fir
            glVertex3i( i   *TILE_RES, j*TILE_RES, 0);  'glVertex
            glNormal3f(0.0, 0.0, 1.0);  'glNormal3f' is deprecated: fir
            glVertex3i((i+1)*TILE_RES, j*TILE_RES, 0);  'glVertex
        }
        glEnd();  'glEnd' is deprecated: first deprecated in macOS 1
    }
    glPopMatrix();  'glPopMatrix' is deprecated: first deprecated in macOS
```

Far Wall: No rotation was carried out. It translates to the far edge and faces the -z axis. Two routes are available. The first is the texture path, which enables textures and merely draws a single, large "GL_QUADS" face mapped with messi.png when "showTextures true" is set. The other is the no-texture path, which separates the wall into panels while leaving a rectangular opening for a sliding glass window. The pre-defined "GLASS_WIDTH, GLASS_HEIGHT, GLASS_ELEV" constants are used by hole logic to skip panels that are above, below, left, and right of the opening. The translucent pane is rendered later by drawGlass(), which skips any geometry that remains in the hole.

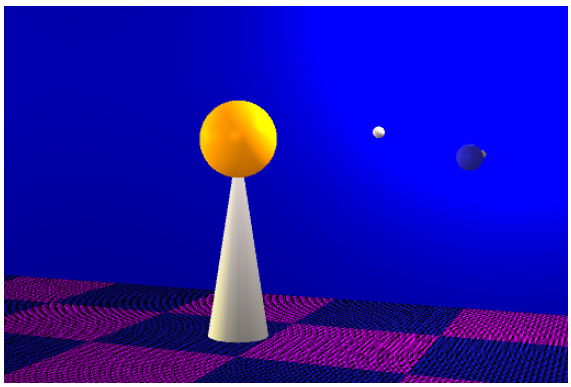
```
void drawWalls()
    // draw far wall
    glMatrixMode(GL_MODELVIEW);  'glMatrixMode' is deprecated: first deprecated in macOS 10.14
    glPushMatrix();  'glPushMatrix' is deprecated: first deprecated in macOS 10.14 - OpenGL API d

    // rotate to face wall
    // glRotated(0.0, 0.0, 1.0, 0.0);

    // move to lower left corner
    glTranslated(ROOM_WIDTH/-2.0, FLOOR_LEVEL, ROOM_LENGTH/-2.0);  'glTranslated' is

    if (showTextures) {
        glEnable(GL_TEXTURE_2D);  'glEnable' is deprecated: first deprecated in macOS 10.14 - O
        glBindTexture(GL_TEXTURE_2D, pix[0]->id);  // skyline3.png  'glBindTexture
        printf("Binding messi texture ID %u on far wall\n", pix[0]->id);

        GLfloat const texColorA[4] = {1.0, 1.0, 1.0, 1.0};
        GLfloat const texColorD[4] = {1.0, 1.0, 1.0, 1.0};
        GLfloat const texColorS[4] = {1.0, 1.0, 1.0, 1.0};

        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, texColorA);  'glMaterialfv' is de
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, texColorD);  'glMaterialfv' is de
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, texColorS);  'glMaterialfv' is d
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 0.0f);  'glMaterialf' is deprecat

        glBegin(GL_QUADS);  'glBegin' is deprecated: first deprecated in macOS 10.14 - OpenGL A
            glTexCoord2f(0.0f, 0.0f); glVertex3f(0.0f,              0.0f, 0.0f);
            glTexCoord2f(1.0f, 0.0f); glVertex3f(ROOM_WIDTH,        0.0f, 0.0f);
            glTexCoord2f(1.0f, 1.0f); glVertex3f(ROOM_WIDTH, ROOM_HEIGHT, 0.0f);
            glTexCoord2f(0.0f, 1.0f); glVertex3f(0.0f,       ROOM_HEIGHT, 0.0f);
        glEnd();  'glEnd' is deprecated: first deprecated in macOS 10.14 - OpenGL API de

        glDisable(GL_TEXTURE_2D);  'glDisable' is deprecated: first deprecated in macOS 10.14
    } else {
        // fallback: draw untextured wall
        for (int i = 0; i < ROOM_WIDTH / TILE_RES; ++i) {
            glBegin(GL_QUAD_STRIP);  'glBegin' is deprecated: first deprecated in macOS 10.14
            for (int j = 0; j <= ROOM_HEIGHT / TILE_RES; ++j) {
                glNormal3f(0.0, 0.0, 1.0);  'glNormal3f' is deprecated: first deprecated in
                glVertex3i(i * TILE_RES, j * TILE_RES, 0);  'glVertex3i' is deprecated
                glVertex3i((i + 1) * TILE_RES, j * TILE_RES, 0);  'glVertex3i' is de
            }
            glEnd();  'glEnd' is deprecated: first deprecated in macOS 10.14 - OpenGL API de
        }
```

This is a good design because it uses grid math only. No hard-coded vertex lists which makes the room scalable by changing four constants. Also, once each face is in its own local frame, vertex code stays trivial “(i*TILE_RES, j*TILE_RES, 0)”. Lastly and most importantly, each surface’s helper can be modified (add bump mapping, different tile size etc.) without touching the others.

Sculptures Of The Museum

In the museum, we have 5 sculptures. Each sculpture is unique to itself. They have different appearances and different functionalities. These are the sculptures in the museum:

- **Sculpture 1 (Mini Solar System Model):** A scaled-down solar-system model shows the Sun, Mercury, and an Earth-Moon pair orbiting on markedly elliptical paths. Visitors can see that Mercury's motion is quick while Earth and its Moon have a smoother and wider motion.



In drawSculpture1(), "glTranslated((ROOM_WIDTH/2)-768, …)" sets the pedestal's world position. A cone-shaped stand is formed with a "GL_TRIANGLE_FAN". Then the Sun is drawn by "gluSphere(quadric, 128, 60, 40)" rendered with a gold-orange material set "(sunA/D/S)". Earth/Moon and Mercury each get their own "glPushMatrix()" so their orbits can be expressed as child transforms. "glTranslated(earthDist * sin(earthTheta), …, earthDist * -cos(earthTheta))" places Earth on its elliptical path; another translate inside that moves the Moon.

```
void drawSculpture()
{
    int i;

    // Material properties
    GLfloat coneA[]    = {0.33, 0.33, 0.33, 1.0};
    GLfloat coneD[]    = {0.78, 0.78, 0.78, 1.0};
    GLfloat coneS[]    = {0.90, 0.90, 0.90, 1.0};

    GLfloat sunA[]     = {0.6, 0.4, 0.1, 1.0};
    GLfloat sunD[]     = {0.8, 0.6, 0.1, 1.0};
    GLfloat sunS[]     = {1.0, 0.8, 0.1, 1.0};

    GLfloat earthA[]   = {0.1, 0.1, 0.4, 1.0};
    GLfloat earthD[]   = {0.1, 0.1, 0.6, 1.0};
    GLfloat earthS[]   = {0.1, 0.1, 0.8, 1.0};

    GLfloat moonA[]    = {0.3, 0.3, 0.3, 1.0};
    GLfloat moonD[]    = {0.8, 0.8, 0.8, 1.0};
    GLfloat moonS[]    = {0.0, 0.0, 0.0, 1.0};

    GLfloat mercuryA[] = {0.3, 0.3, 0.3, 1.0};
    GLfloat mercuryD[] = {0.8, 0.8, 0.8, 1.0};
    GLfloat mercuryS[] = {0.0, 0.0, 0.0, 1.0};

    // Start drawing
    glPushMatrix();

    glTranslated((ROOM_WIDTH / 2.0) - 768.0, 0.0, (ROOM_LENGTH / 2.0) - (2.0 * ROOM_LENGTH / 5.0));

    // Stand base
    setMaterial(coneA, coneD, coneS, 27.8f);
    glBegin(GL_TRIANGLE_FAN);
        glNormal3f(0, 1.0, 0.0);
        glVertex3d(0.0, 0.0, 0.0);
        for (i = 0; i <= TILE_RES; ++i) {
            double angle = i * 2.0 * M_PI / TILE_RES;
            glNormal3f(sin(angle), 0.0, cos(angle));
            glVertex3d(100.0 * sin(angle), FLOOR_LEVEL, 100.0 * cos(angle));
        }
    glEnd();
```

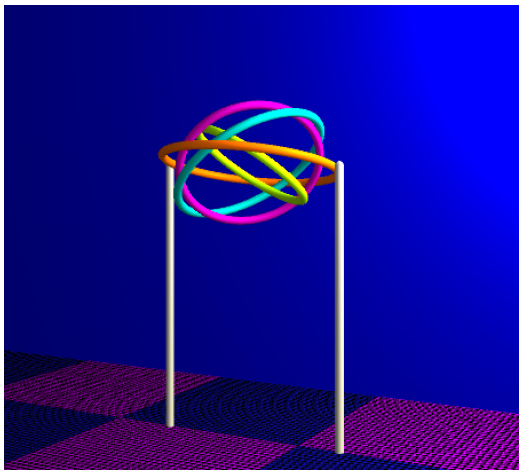
```
void drawSculpture1()
    // Sun
    setMaterial(sunA, sunD, sunS, 100.0f);
    gluSphere(quadric, 128.0, 60, 40);
    glRotated(5.0, 0.0, 0.0, 1.0);  // Tilt for aesthetics

    // Earth + Moon
    glPushMatrix();
        glTranslated(earthDist * sin(earthTheta), 0.0, earthDist * -cos(earthTheta));
        setMaterial(earthA, earthD, earthS, 100.0f);
        gluSphere(quadric, 32.0, 35, 25);

        glTranslated(moonDist * sin(moonTheta), 0.0, moonDist * -cos(moonTheta));
        setMaterial(moonA, moonD, moonS, 1.0f);
        gluSphere(quadric, 10.0, 20, 15);
    glPopMatrix();

    // Mercury
    glPushMatrix();
        glTranslated(mercuryDist * sin(mercuryTheta), 0.0, mercuryDist * -cos(mercuryTheta));
        setMaterial(mercuryA, mercuryD, mercuryS, 1.0f);
        gluSphere(quadric, 20.0, 20, 15);
    glPopMatrix();

    glPopMatrix();
}
```

The updateSculpture1() runs every timer tick. It computes a crude inverse-square velocity. "earthVelocity = (75000 / (earthDist²)) – (π/220)" then advances "earthTheta += earthVelocity * delta". The varying "earthDist" is recalculated with the ellipse

equation “P / (1+e cosθ)”, giving true perihelion/aphelion motion without key-frames. Each body gets its own ambient/diffuse/specular tuple; the Moon and Mercury use low specular to stay matte, while the Sun and stand are shiny.

```
// update sculpture1 animation
void updateSculpture1()
{
    // Orbital constants
    const GLdouble EARTH_P = 350.0;
    const GLdouble EARTH_E = 0.75;

    const GLdouble MERCURY_P = 250.0;
    const GLdouble MERCURY_E = 0.58;

    GLdouble delta = (ANI_RATE / 200.0) * speedMultiplier;

    // Earth update
    GLdouble earthVelocity = (75000.0 / (earthDist * earthDist)) - (M_PI / 220.0);
    earthTheta += earthVelocity * delta;
    earthTheta  = fmod(earthTheta, 2.0 * M_PI);
    earthDist   = EARTH_P / (1 + EARTH_E * cos(earthTheta));

    // Moon update
    moonTheta += (M_PI / 6.0) * delta;
    moonTheta  = fmod(moonTheta, 2.0 * M_PI);

    // Mercury update
    GLdouble mercuryVelocity = (60000.0 / (mercuryDist * mercuryDist)) - (M_PI / 220.0);
    mercuryTheta += mercuryVelocity * delta;
    mercuryTheta  = fmod(mercuryTheta, 2.0 * M_PI);
    mercuryDist   = MERCURY_P / (1 + MERCURY_E * cos(mercuryTheta));
}
```

- **Sculpture 2 (Nested Rotating Tori):** Four colored doughnuts spin around different axes and at four different speeds, forming a hypnotic kinetic sculpture anchored to the floor by two vertical support posts.



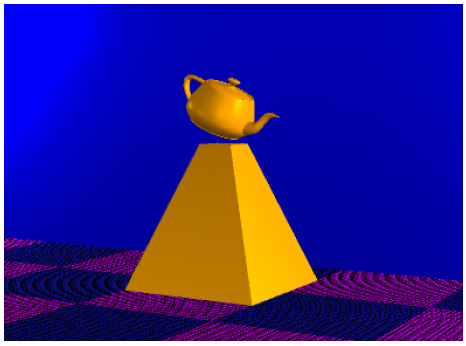
The drawSculpture2() translates the whole stack to the left-hand wing of the room, then rotates the local frame 90° about the Y-axis so the tori lie flat. A loop iterates “i = 0…3;”. The loop first sets a material triple from “colors[i*3+…]”. Then, it applies “glRotated(diskRot[i], (i%2), (i%2^1), 0)”. Even-numbered rings spin around X, odd around Y. Lastly it calls “glutSolidTorus(10, 210-20*i, 20, 50).

```
void drawSculpture2()
{
    const GLfloat colors[][4] = {
        {0.4, 0.2, 0.0, 1.0}, {0.8, 0.4, 0.0, 1.0}, {1.0, 0.5, 0.0, 1.0},    // Torus 1 - Orange
        {0.0, 0.3, 0.3, 1.0}, {0.0, 0.6, 0.6, 1.0}, {0.0, 0.9, 0.9, 1.0},    // Torus 2 - Teal
        {0.3, 0.0, 0.3, 1.0}, {0.6, 0.0, 0.6, 1.0}, {0.9, 0.0, 0.9, 1.0},    // Torus 3 - Magenta
        {0.3, 0.4, 0.0, 1.0}, {0.6, 0.8, 0.1, 1.0}, {0.9, 1.0, 0.3, 1.0},    // Torus 4 - Yellow-Green
        {0.3, 0.3, 0.3, 1.0}, {0.6, 0.6, 0.6, 1.0}, {0.8, 0.8, 0.8, 1.0}     // Support cylinders/spheres (neutral gray)
    };

    glMatrixMode(GL_MODELVIEW);
        glPushMatrix();
        glTranslated(-ROOM_WIDTH / 2.0 + 512, 0.0, ROOM_LENGTH / 2.0 - (2.0 * ROOM_LENGTH / 8.0));
        glRotated(90.0, 0.0, 1.0, 0.0);

        glPushMatrix();
        for (int i = 0; i < 4; ++i) {
            glRotated(diskRot[i], (i % 2 == 0), (i % 2 == 1), 0.0);

            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,   colors[i * 3 + 0]);
            glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,   colors[i * 3 + 1]);
            glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  colors[i * 3 + 2]);
            glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100.0f);

            glDisable(GL_CULL_FACE);
            glutSolidTorus(10.0, 210.0 - 20 * i, 20, 50);
            glEnable(GL_CULL_FACE);
        }
        glPopMatrix();

        for (int side = -1; side <= 1; side += 2) {
            glPushMatrix();
            glTranslated(side * 230.0, 0.0, 0.0);
            glRotated(90.0, 1.0, 0.0, 0.0);

            glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,   colors[12]);
            glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,   colors[13]);
            glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  colors[14]);
            glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100.0f);

            gluCylinder(quadric, 10.0, 10.0, -1.0 * FLOOR_LEVEL, 20, 80);
            gluSphere(quadric, 10.0, 10, 15);
            glPopMatrix();
        }

        glPopMatrix();
    }
```

The updateSculpture2() keeps an array diskRot[4]. Every tick:
“diskRot[i] += rotationSpeeds[i] * delta;” where “speeds = {5, 15, 25, 35} °/frame”. A “fmod” ensures angles stay within 0-360. Two cylinders “gluCylinder” and spheres anchor the stack to the floor so it feels mechanically valid.

```
// update sculpture2 animation
void updateSculpture2() {
    const GLfloat rotationSpeeds[] = {5.0, 15.0, 25.0, 35.0};
    GLfloat delta = (ANI_RATE / 200.0) * speedMultiplier;

    for (int i = 0; i < 4; ++i) {
        diskRot[i] += rotationSpeeds[i] * delta;
        diskRot[i] = fmod(diskRot[i], 360.0);
    }
}
```

- **Sculpture 3 (Golden Pouring Teapot)**: A classic Utah teapot, perched on a pedestal, gently tilts back and forth. When it pours forward, a short water-pour sound plays. The sound depends on the users demand.



The drawSculpture3() draws a truncated-pyramid stand via “drawFrustum()”, then rotates the teapot 90° about Y to face visitors, and adds a “-teapotTiltAngle” roll about

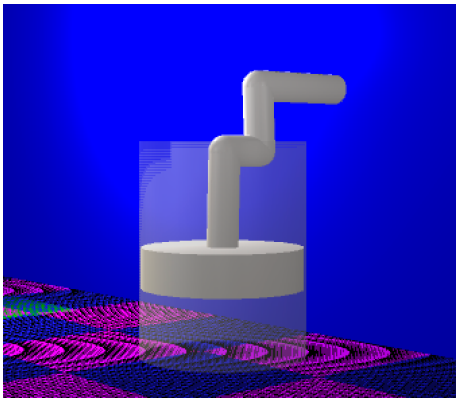
Z for the pouring motion. A single gold-like material “ambient/diffuse/specular = 0.33/0.78/0.99” is applied to the stand and pot.

```
void drawSculpture3() {
    const GLfloat ambient[]  = {0.33, 0.22, 0.03, 1.0};
    const GLfloat diffuse[]  = {0.78, 0.57, 0.11, 1.0};
    const GLfloat specular[] = {0.99, 0.91, 0.81, 1.0};
    const GLfloat shininess  = 100.0f;

    // Set material properties for entire sculpture
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT,   ambient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE,   diffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR,  specular);
    glMaterialf( GL_FRONT_AND_BACK, GL_SHININESS, shininess);

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

        // Position sculpture in scene
        GLdouble x = -ROOM_WIDTH / 2.0 + 512;
        GLdouble z = ROOM_LENGTH / 2.0 - 4.0 * ROOM_LENGTH / 8.0;
        glTranslated(x, 0.0, z);

        // Draw base stand
        glPushMatrix();
        glTranslated(0.0, FLOOR_LEVEL, 0.0);
        drawFrustum(512.0, 128.0, 512.0);
        glPopMatrix();

        // Rotate and draw teapot
        glRotated(90.0, 0.0, 1.0, 0.0);  // keep this for orientation
        glRotated(-teapotTiltAngle, 0.0, 0.0, 1.0); // tilt forward/backward
        glDisable(GL_CULL_FACE);
        glutSolidTeapot(128.0);
        glEnable(GL_CULL_FACE);

    glPopMatrix();
}
```

The updateSculpture3() increments “teapotTiltAngle” by “tiltSpeed = 2°/tick”, reverses at 45°, and toggles “teapotPouringForward”. When “playPourSound” is “true” and the teapot first enters its forward half-cycle, system("afplay pour.wav &") fires. A “soundPlayed” flag prevents repeats until the next cycle.

```
void updateSculpture3()
{
    const float tiltSpeed = 2.0f;
    const float maxTilt = 45.0f;

    if (teapotPouringForward) {
        teapotTiltAngle += tiltSpeed;

        // 🔊 Only play sound if user enabled it with 'p'
        if (playPourSound && !soundPlayed && teapotTiltAngle >= tiltSpeed) {
            system("afplay pour.wav &");  // async sound (macOS)
            soundPlayed = true;
        }

        if (teapotTiltAngle >= maxTilt) {
            teapotTiltAngle = maxTilt;
            teapotPouringForward = false;
        }
    } else {
        teapotTiltAngle -= tiltSpeed;
        if (teapotTiltAngle <= 0.0f) {
            teapotTiltAngle = 0.0f;
            teapotPouringForward = true;
            soundPlayed = false;  // reset for next pour
        }
    }
}
```

- **Sculpture 4 (Piston & Crank Assembly):** This sculpture is a transparent acrylic cylinder that reveals a working piston driven by a rotating crankshaft and connecting rod, a textbook demonstration of reciprocating motion.



The drawSculpture4() builds a wall-mount “gluCylinder” and a crankshaft sphere. The crank’s Z-length equals “crankRadius”. Its rotation uses “-crankTheta * 180/π + 90” so 0 rad = “pointing down”. After translating by “-pistHeight”, a 256-pixel-radius “gluCylinder” plus two end-disks form the piston. The push-rod orientation is “asin(crankRadius sinθ / rodLength)” so the rod always meets the crank pin. A separate pass draws a tall cylinder with %30 alpha, blending controlled by “blockAmbient/Diffuse/Specular”.

```
void drawSculpture4()
{
    // --- Define Material Properties ---
    const GLfloat metalAmbient[]  = {0.4, 0.4, 0.4, 1.0};
    const GLfloat metalDiffuse[]  = {0.6, 0.6, 0.6, 1.0};
    const GLfloat metalSpecular[] = {0.6, 0.6, 0.6, 1.0};

    const GLfloat blockAmbient[]  = {0.4, 0.4, 0.4, 0.30};
    const GLfloat blockDiffuse[]  = {0.4, 0.4, 0.4, 0.30};
    const GLfloat blockSpecular[] = {1.0, 1.0, 1.0, 0.30};

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

    // --- Position the Sculpture ---
    glTranslated((ROOM_WIDTH / 2.0) - 512, 200.0, (ROOM_LENGTH / 2.0) - (3.0 * ROOM_LENGTH / 5.0));

    // --- Explosion Effect (Currently Disabled) ---
    if (0) {
        glPushMatrix();

        const GLfloat redA[] = {0.4, 0.0, 0.0, 1.0};
        const GLfloat redD[] = {0.8, 0.0, 0.0, 1.0};
        const GLfloat redS[] = {0.9, 0.0, 0.0, 1.0};

        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, redA);
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, redD);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, redS);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100.0f);

        glTranslated(0.0, -rodLength - 420.0, 0.0);

        if (fabs(crankTheta) < (40.0 * M_PI / 180.0))
            glScaled(1.0, 200.0 / pistHeight + 0.1, 1.0);
        else
            glScaled(1.0, 0.9 - 200.0 / pistHeight + 0.1, 1.0);

        gluSphere(quadric, 256.0, 20, 30);
        glPopMatrix();
    }

    // --- Metallic Look for Piston Assembly ---
    glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, metalAmbient);
    glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, metalDiffuse);
    glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, metalSpecular);
    glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100.0f);
```

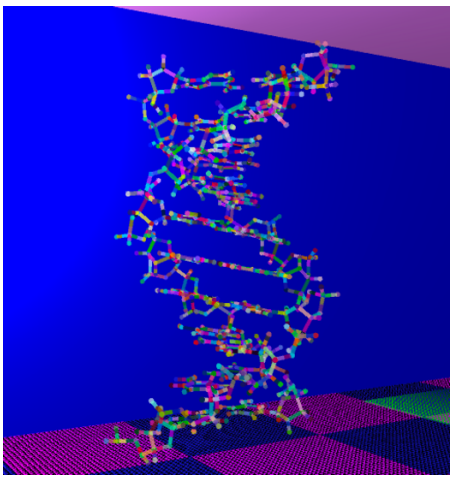
```
void drawSculpture4()
    // --- Draw Main Piston Assembly ---
    glPushMatrix();

    // Crank Shaft Wall Mount
    glPushMatrix();
        glTranslated(150.0, 0.0, 0.0);
        glRotated(90.0, 0.0, 1.0, 0.0);
        gluSphere(quadric, 50.0, 20, 30);
        gluCylinder(quadric, 50.0, 50.0, 362.0, 20, 30);
    glPopMatrix();

    // Rotating Crank
    glPushMatrix();
        glTranslated(150.0, 0.0, 0.0);
        glRotated(-crankTheta * 180.0 / M_PI + 90.0, 1.0, 0.0, 0.0);
        gluCylinder(quadric, 50.0, 50.0, crankRadius, 20, 30);
        glTranslated(0.0, 0.0, crankRadius);
        gluSphere(quadric, 50.0, 20, 30);
    glPopMatrix();

    // Apply piston height and orientation
    glTranslated(0.0, -pistHeight, 0.0);
    glRotated(90.0, 1.0, 0.0, 0.0);

    // Main piston
    gluCylinder(quadric, 256.0, 256.0, 128.0, 20, 30);

    // Top of piston
    glRotated(180.0, 1.0, 0.0, 0.0);
    gluDisk(quadric, 0.0, 256.0, 20, 30);

    // Push Rod Mechanism
    glPushMatrix();
        glRotated(asin(crankRadius * sin(crankTheta) / rodLength) * 180.0 / M_PI, 1.0, 0.0, 0.0);
        gluSphere(quadric, 50.0, 20, 30);
        gluCylinder(quadric, 50.0, 50.0, rodLength, 20, 30);

        // Joint to crankshaft
        glTranslated(0.0, 0.0, rodLength);
        glRotated(90.0, 0.0, 1.0, 0.0);
        gluSphere(quadric, 50.0, 20, 30);
        gluCylinder(quadric, 50.0, 50.0, 150.0, 20, 30);
    glPopMatrix();
```

```
void drawSculpture4()

    // Bottom cap of piston
    glTranslated(0.0, 0.0, -128.0);
    glRotated(-180.0, 1.0, 0.0, 0.0);
    gluDisk(quadric, 0.0, 256.0, 20, 30);

    glPopMatrix(); // End of piston assembly

    // --- Draw Transparent Block Enclosure ---
    glPushMatrix();
        glTranslated(0.0, FLOOR_LEVEL - 200, 0.0);
        glRotated(-90.0, 1.0, 0.0, 0.0);

        glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT, blockAmbient);
        glMaterialfv(GL_FRONT_AND_BACK, GL_DIFFUSE, blockDiffuse);
        glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, blockSpecular);
        glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 100.0f);

        glDisable(GL_CULL_FACE);
        gluCylinder(quadric, 260.0, 260.0, 670.0, 60, 80);
        glEnable(GL_CULL_FACE);
    glPopMatrix();

    glPopMatrix(); // End of sculpture
}
```

The updateSculpture4() steps “crankTheta += 35° · delta”, then solves the classic slider-crank equation which is “pistHeight = R cosθ + √(L² − R² sin²θ)”.

```
void updateSculpture4()
{
    // Increment crank angle based on animation rate and speed multiplier
    crankTheta += (35.0 * (M_PI / 180.0)) * (ANI_RATE / 200.0) * speedMultiplier;
    crankTheta = fmod(crankTheta, 2.0 * M_PI);

    // Enable burn effect if crank is near vertical
    showBurn = fabs(crankTheta) < (90.0 * (M_PI / 180.0));

    // Update piston height using trigonometric crank-rod relationship
    double sinTheta = sin(crankTheta);
    double cosTheta = cos(crankTheta);
    pistHeight = crankRadius * cosTheta + sqrt(rodLength * rodLength - crankRadius * crankRadius * sinTheta * sinTheta);
}
```

- **Sculpture 5 (DNA Double Helix)**: A spiral DNA ladder, two intertwined backbones with rungs and stands near the entrance. The visitor can hide or reveal it with the “h” key.



The drawSculpture5() immediately returns “if showHelix == false”, letting the key toggle act as an inexpensive on/off switch. The helix is rotated –95° about X so it rises vertically, scaled *35, and translated to its corner location.

```
void drawSculpture5()
{
    if (!showHelix) return;

    glMatrixMode(GL_MODELVIEW);
    glPushMatrix();

        // Position the sculpture in the scene
        glTranslated(
            (ROOM_WIDTH / -2.0) + 512.0,
            0.0,
            (ROOM_LENGTH / 2.0) - (6.0 * ROOM_LENGTH / 8.0)
        );

        // Rotate and scale the helix
        glRotated(-95.0, 1.0, 0.0, 0.0);
        glScaled(35.0, 35.0, 35.0);

        // Draw the actual structure
        drawDoubleHelix();

    glPopMatrix();
}
```

The drawDoubleHelix() (from doubleHelix.c) procedurally computes two sinusoidal backbones and pairs of quad-strip rungs.Due to the self-contained helix, it requires no per-frame update. Any future spin would go into updateSculpture5().

```
void drawDoubleHelix()
{
    // same colors
    srand(779);

    // each call updated modelview
    _glMatrixMode(GL_MODELVIEW);  'glMatrixMode' is depr

    // Atoms and NICS cubes
    drawMolicule(-0.808, -8.873, -17.29, 0.23 );
    drawMolicule(-0.196, -10.198, -17.07, 0.23 );
    drawMolicule(-2.029, -8.583, -16.5, 0.23 );
    drawMolicule(0.27, -7.725, -17.04, 0.23 );
    drawMolicule(4.189, -7.682, -15.46, 0.23 );
    drawMolicule(2.252, -5.41, -15.68, 0.23 );
    drawMolicule(3.108, -4.88, -15.262, 0.23 );
    drawMolicule(2.035, -6.74, -14.97, 0.23 );
    drawMolicule(2.42, -6.732, -13.95, 0.23 );
    drawMolicule(2.818, -7.701, -15.85, 0.23 );
    drawMolicule(2.437, -8.72, -15.781, 0.23 );
    drawMolicule(2.608, -7.128, -17.25, 0.23 );
    drawMolicule(3.47, -7.426, -17.847, 0.23 );
    drawMolicule(1.364, -7.578, -17.98, 0.23 );
    drawMolicule(1.104, -6.836, -18.735, 0.23 );
    drawMolicule(2.53, -5.682, -17.04, 0.23 );
    drawMolicule(-1.645, -0.802, -15.35, 0.23 );
    drawMolicule(-1.755, 0.217, -15.252, 0.23 );
```

Interactive Features

In our project, the museum has various different animations, functionalities that can be controlled by the user. All interactivity is controlled through a single GLUT keyboard callback (keyDown()) plus the navigator's mouse/arrow logic. keyDown() does nothing more than flip a flag or tweak a scalar and those variables are read each frame by the draw or animate routines, guaranteeing that the render loop remains clean and stateless. Since each feature is reduced to "flip a flag, request redisplay," the interactive layer is easy to extend, the extensions can be done via adding a new effect is usually one boolean, one if and a sprinkle of math in the next animation cycle. These are the current features that we have in our museum:

```
=== Museum Summary ===
• Sculpture 1: Mini Solar System model with Sun, Earth, Moon, Mercury.
• Sculpture 2: Rotating multi-colored torus disks.
• Sculpture 3: Golden teapot tilting animation.
• Sculpture 4: Mechanical piston and crank assembly.
• Sculpture 5: DNA double helix structure.
• Press 't': Image on wall and ceiling
• Press '1-8': Lighting of the museum.
• Press 'Arrow Keys': Move in the museum.
• Press 'd': Little up and down movement.
• Press 'i': Information about the museum.
• Press 'p': Play teapot pouring sound.
• Press 's': Double animation speed.
• Press 'a': Freeze/unfreeze animations.
• Press 'm': Music for museum.
• Press 'q': Quitting the museum.
• Press 'h': Show/Disappear Sculptire 5.
• Press '+': Zoom in.
• Press '-': Zoom out.
• Press 'j': Jump in the museum.
======================
```

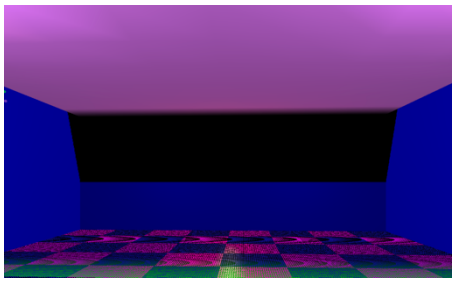
- **Digits "1-8" (Light Toggles):** Converts the ASCII digit to an index, then calls :

  "if (glIsEnabled(GL_LIGHT0+idx)) glDisable(GL_LIGHT0+idx);
  else glEnable (GL_LIGHT0+idx);"

  The fixed-function engine handles the rest. No redraw logic is needed beyond glutPostRedisplay().

- **Key "t" (Texture Mapping on/off):** Flips showTextures. Each draw routine guards its texture binds:

  "if (showTextures) {
  glEnable(GL_TEXTURE_2D);
  …
  } else
  glDisable(GL_TEXTURE_2D);"



- **Key “h” (DNA double helix visibility):** Toggles showHelix. drawSculpture5() begins with:

  “if (!showHelix)
  return;”

  No extra state handling is required.

- **Key “m” (Backgorund Music):** If music is off, fork()s and execlp()s afplay asynchronously.

  If on, sends SIGKILL to the stored musicPID.

  Audio is kept outside the render loop, so GPU timing is unaffected.

- **Key “a” (Freeze / unfreeze animation**): Sets frozen. The timer callback animate() early-outs when frozen is true, pausing all sculpture updates with one boolean.

- **Key “p” (Teapot pouring sound):** Flips playPourSound; updateSculpture3() plays pour.wav the first time the teapot crosses its forward-tilt threshold, guarded so the sample fires only once per cycle.

- **Key “q” (Quit museum):** cleanUpAndQuit() is called. It iterates through the "pix[] array," nulling the pointers and "free()"ing each decoded PNG buffer. When the process ends, any background music that is playing is implicitly stopped. calls "exit(ALL_IS_WELL);" to make sure the application releases the OpenGL context and gives the operating system a success message.

- **Key “s” (Double/Normal speed):** Toggles speedMultiplier between 1.0 and 2.0. Every delta computation in the update functions is multiplied by this scalar, giving a global “fast-forward” effect.

- **Key “i” (Guidance of museum):** Gives a brief information about the sculptures and the key functionalities in the output bar. Can be extended or decreased by the user.

- **Keys “Arrows”:** Set smooth-motion booleans consumed by navSmoothMotion(), which pulses movement/turn timers every 75 ms.

- **Key “j” (Jumping):** starts a parabolic jump (decrementing jumpUnit each tick).

- **Key “d” (Up and Down Movement): T**oggles duck height.

- **Key “+/-“ (Zoom in/out):** Call navZoom(). Mouse L/M/R buttons switch modes (strafe, zoom, look) with pointer warping for infinite mouselook.

```
void keyDown(unsigned char key, int x, int y)
    if (isdigit(key)) {
        int keyDigit = key - '0';
        if (keyDigit >= 1 && keyDigit <= 8) {
            int index = keyDigit - 1;
            if (glIsEnabled(lights[index]))
                glDisable(lights[index]);
            else
                glEnable(lights[index]);

            glutPostRedisplay();
        }
        return;
    }

    switch (key) {
        case 'a':
            frozen = !frozen;
            glutPostRedisplay();
            break;

        case 'f':
            frozen = true;
            if (!gameMode) {
                if (glutGameModeGet(GLUT_GAME_MODE_POSSIBLE)) {
                    gameWindowID = glutGetWindow();
                    glutEnterGameMode();

                    navInitDisplay();
                    navInitCallBacks();
                    initCallBacks();
                    initLighting();
                    initTextures();

                    gameMode = true;
                } else {
                    fprintf(stderr, "Full screen mode is not available.\n");
                }
            } else {
                glutLeaveGameMode();
                glutSetWindow(gameWindowID);

                navInitDisplay();
                navInitCallBacks();
                initCallBacks();
                initLighting();
```

```
void keyDown(unsigned char key, int x, int y)
        case 'h':
            showHelix = !showHelix;
            glutPostRedisplay();
            break;

        case 'k':
            capture = !capture;
            break;

        case 'q':
            cleanUpAndQuit();
            break;

        case 't':
            showTextures = !showTextures;
            glutPostRedisplay();
            break;

        case 'm':
            if (!musicPlaying) {
                musicPID = fork();
                if (musicPID == 0) {
                    // Child process: play music
                    execlp("afplay", "afplay", "background.wav", (char *)NULL);
                    exit(1); // If execlp fails
                }
                musicPlaying = true;
            } else {
                if (musicPID > 0) {
                    kill(musicPID, SIGKILL);
                    musicPID = -1;
                }
                musicPlaying = false;
            }
            break;
        case 'i':
            printf("\n=== Museum Summary ===\n");
            printf("* Sculpture 1: Mini Solar System model with Sun, Earth, Moon, Mercury.\n");
            printf("* Sculpture 2: Rotating multi-colored torus disks.\n");
            printf("* Sculpture 3: Golden teapot tilting animation.\n");
            printf("* Sculpture 4: Mechanical piston and crank assembly.\n");
            printf("* Sculpture 5: DNA double helix structure.\n");
            printf("* Press 'a': Freeze the museum.\n");
            printf("* Press 't': Image on wall and ceiling\n");
            printf("* Press '1-8': Lighting of the museum.\n");
```

```
void keyDown(unsigned char key, int x, int y)

        case 'w':
            glassIsOpening = !glassIsOpening;
            break;
        case 's':
            speedMultiplier = (speedMultiplier == 1.0) ? 2.0 : 1.0;
            break;
        case 'p':
            playPourSound = !playPourSound;
            break;
        case 'e':
            printf("DEBUG: 'e' key pressed — starting camera shake.\n");
            fflush(stdout);  // ensures output is printed immediately
            cameraShaking = true;
            shakeFrame = 0;
            glutPostRedisplay();  // force redraw for instant feedback
            break;
        default:
            break;
    }
}
```

Future Implementations

For the museum, we think that we have implemented a decent amount changes. The museum has interesting sculptures and interactive features that can quite excite the user. However, we think that this museum can be improved with some modifications and features added. For example, we are planning to add a full screen mode. This will replace the current windowed experience. It will finish the GLUT game-mode hand-off, remove OS chrome and deepen immersion. Also, a teleport-to-sculpture feature can let visitors jump instantly to any artwork, it is useful for demonstrations or accessibility. The most important upgrade that we are planning to work on is an “outside-to-inside” entry sequence. The scene will start in the outdoor courtyard, automatically animate the glass door sliding open, and guide the camera through the threshold before handing control to the user, turning the static lobby into a narrative prologue. There are also some features that we started implementing but could not finish such as the camera shake. With these additions and modifications to the museum, we are aiming to increase the attention and interaction of the users to the museum. We hope that it will increase their satsifaction as well.

Conclusion

Our project started as a idea of can we implement a “Interactive 3-D Virtual Museum” built with nothing more than modern OpenGL and GLUT frameworks. Is it possible to achieve a fully navigable, visually rich gallery with OpenGL? We had this idea because we thought that it would be a decent two person project that includes the knowledge we gained form the couse and show our creativity as well. In the end, we had codebase that assembles floor to ceiling geometry on the fly, bathes it in an eight-light Phong rig, textures key surfaces and animates five procedurally driven sculptures that span astronomy, kinematics and molecular biology. A lightweight navigation engine grants smooth first person control, jump and duck motion, zoom while small hooks such as toggle-able textures, per-lamp switches and optional music demonstrate how interactive features can be layered onto a fixed function core without bloating complexity. Just as important as the visuals, the code itself remains transparent. Every subsystem (texture loading, lighting, animation) can be read, modified or replaced in isolation. This makes the museum a living textbook for graphics students.In our opinion, with the future implementations that we will provide, this project will has a wide future and can be developed into a much more comprehensive and forward looking project or new project ideas by experts.

References

- https://www.youtube.com/watch?v=aYIYXrAiiAA.
- https://www.elliottforney.com.