



An example of how the mshell Ecosystem works.

Bouncing Balls Prompt

This is an example of an automated code generation session for solving a practical problem with mshell ecosystem (mshell – AI and mathematics powered shell, mshell Studio Library Manager, mshell Studio Validator, mshell Editor (Edi), . During the session, you adjust and modify the application task throw prompt until you achieve the desired result.

/home/igor > Sent to mshell (315 bytes)

Received from GUI editor: ### ollama1exec “write C program with OpenGL that creates dynamic picture of 3 bouncing all time balls with gold, silver and cooper cover color. Balls needs jump all time without stop and you need create picture with good resolution 1024x768 pixels. Why your balls are stoping? They need to jump on high of all screen.”

They were stopping because I used “realistic” physics with gravity and energy loss. Every bounce lost a bit of speed, so over time the velocity decayed toward zero. Also, I kept the motion in a small region, not full-screen height.

Below is a C + OpenGL (GLUT) program that:

- Opens a 1024×768 window.
- Draws 3 balls (gold, silver, copper).
- Makes each ball bounce vertically from near the bottom to near the top of the view.
- Balls **never stop** and always bounce with the same amplitude (we use a periodic function instead of decaying gravity).

You can compile it similarly to before:

- Linux: gcc bouncing_fullscreen.c -lGL -lGLU -lglut -lm -o bouncing_fullscreen
- Windows (MinGW): gcc bouncing_fullscreen.c -lopengl32 -lglu32 -lglut32 -lm -o bouncing_fullscreen

```
/*  
* 3 Bouncing Balls (Gold, Silver, Copper)
```

```

* Window size: 1024 x 768
* Balls bounce forever from near bottom to near top of view.
* Uses GLUT + fixed-function OpenGL.
*/

#include <GL/glut.h>
#include <math.h>

/* Window size */
const int WIN_WIDTH  = 1024;
const int WIN_HEIGHT = 768;

/* Ball structure */
typedef struct {
    float x;           /* fixed x position */
    float radius;
    float baseY;      /* vertical center of motion */
    float amplitude;  /* how high above/below baseY they move */
    float speed;      /* angular speed for sine wave */
    float phase;      /* phase offset */
    float color[4];   /* RGBA */
} Ball;

Ball balls[3];

/* Global time (seconds) */
float globalTime = 0.0f;

/* Initialize lighting and materials */
void initLighting(void)
{
    glEnable(GL_LIGHTING);
    glEnable(GL_LIGHT0);
    glEnable(GL_DEPTH_TEST);
    glEnable(GL_NORMALIZE);

    GLfloat light_pos[]  = { 1.0f, 1.0f, 2.0f, 0.0f };
    GLfloat light_amb[]  = { 0.15f, 0.15f, 0.15f, 1.0f };
    GLfloat light_diff[] = { 0.9f, 0.9f, 0.9f, 1.0f };
    GLfloat light_spec[] = { 1.0f, 1.0f, 1.0f, 1.0f };

    glLightfv(GL_LIGHT0, GL_POSITION, light_pos);
    glLightfv(GL_LIGHT0, GL_AMBIENT, light_amb);
    glLightfv(GL_LIGHT0, GL_DIFFUSE, light_diff);
    glLightfv(GL_LIGHT0, GL_SPECULAR, light_spec);

    glShadeModel(GL_SMOOTH);
}

```

```

/* Initialize balls so they move over almost full vertical screen range */
void initBalls(void)
{
    /* Roughly map world y from -1.0 to +1.0 to window height */
    /* We want them to bounce from near bottom to near top: */
    float bottom = -0.9f;
    float top     = 0.9f;
    float center  = (bottom + top) * 0.5f;
    float halfRange = (top - bottom) * 0.5f;

    /* We use amplitude a bit less than halfRange to keep inside the view */
    float amplitude = halfRange * 0.9f;

    /* Gold ball */
    balls[0].x          = -0.6f;
    balls[0].radius     = 0.12f;
    balls[0].baseY      = center;
    balls[0].amplitude  = amplitude;
    balls[0].speed      = 2.0f;    /* radians per second */
    balls[0].phase       = 0.0f;
    balls[0].color[0]   = 1.0f;    /* gold-like */
    balls[0].color[1]   = 0.843f;
    balls[0].color[2]   = 0.0f;
    balls[0].color[3]   = 1.0f;

    /* Silver ball */
    balls[1].x          = 0.0f;
    balls[1].radius     = 0.12f;
    balls[1].baseY      = center;
    balls[1].amplitude  = amplitude;
    balls[1].speed      = 2.5f;    /* faster */
    balls[1].phase       = 1.0f;    /* different phase */
    balls[1].color[0]   = 0.75f;
    balls[1].color[1]   = 0.75f;
    balls[1].color[2]   = 0.75f;
    balls[1].color[3]   = 1.0f;

    /* Copper ball */
    balls[2].x          = 0.6f;
    balls[2].radius     = 0.12f;
    balls[2].baseY      = center;
    balls[2].amplitude  = amplitude;
    balls[2].speed      = 1.7f;
    balls[2].phase       = 2.0f;
    balls[2].color[0]   = 0.722f;
    balls[2].color[1]   = 0.451f;
    balls[2].color[2]   = 0.20f;
    balls[2].color[3]   = 1.0f;
}

```

```

/* Reshape callback: projection setup */
void reshape(int w, int h)
{
    if (h == 0) h = 1;
    glViewport(0, 0, w, h);

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    /* Simple perspective */
    gluPerspective(60.0, (double)w / (double)h, 0.1, 10.0);

    glMatrixMode(GL_MODELVIEW);
}

/* Draw ground for reference (not physically used) */
void drawGround(void)
{
    GLfloat mat_amb_diff[] = {0.1f, 0.3f, 0.1f, 1.0f};
    GLfloat mat_spec[]     = {0.1f, 0.1f, 0.1f, 1.0f};
    GLfloat mat_shininess  = 5.0f;

    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
    glMaterialfv(GL_FRONT, GL_SPECULAR,           mat_spec);
    glMaterialf (GL_FRONT, GL_SHININESS,          mat_shininess);

    glNormal3f(0.0f, 1.0f, 0.0f);
    glBegin(GL_QUADS);
        glVertex3f(-2.0f, -1.0f, -2.0f);
        glVertex3f( 2.0f, -1.0f, -2.0f);
        glVertex3f( 2.0f, -1.0f,  2.0f);
        glVertex3f(-2.0f, -1.0f,  2.0f);
    glEnd();
}

/* Draw one ball at a given y computed from sine */
void drawBall(const Ball *b, float t)
{
    /* Compute vertical position: full-range sinusoidal bounce */
    float y = b->baseY + b->amplitude * fabsf(sinf(b->speed * t + b->phase));
    /* fabs(sin) makes it bounce like up/down off the "floor" */

    GLfloat mat_amb_diff[] = { b->color[0], b->color[1], b->color[2], b->
    >color[3] };
    GLfloat mat_spec[]     = { 1.0f, 1.0f, 1.0f, 1.0f };
    GLfloat mat_shininess  = 60.0f;

    glMaterialfv(GL_FRONT, GL_AMBIENT_AND_DIFFUSE, mat_amb_diff);
    glMaterialfv(GL_FRONT, GL_SPECULAR,           mat_spec);
}

```

```

    glMaterialf (GL_FRONT, GL_SHININESS,          mat_shininess);

    glPushMatrix();
        glTranslatef(b->x, y, 0.0f);
        glutSolidSphere(b->radius, 40, 40);
    glPopMatrix();
}

/* Display callback */
void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT);
    glLoadIdentity();

    /* Camera */
    gluLookAt(0.0, 0.0, 2.2,
              0.0, 0.0, 0.0,
              0.0, 1.0, 0.0);

    drawGround();

    drawBall(&balls[0], globalTime);
    drawBall(&balls[1], globalTime);
    drawBall(&balls[2], globalTime);

    glutSwapBuffers();
}

/* Idle: update global time and redraw continuously */
void idle(void)
{
    static int last_ms = 0;
    int ms = glutGet(GLUT_ELAPSED_TIME);
    if (last_ms == 0) last_ms = ms;

    int dt_ms = ms - last_ms;
    last_ms = ms;

    /* Convert to seconds */
    float dt = dt_ms / 1000.0f;
    globalTime += dt;

    /* Keep t in a reasonable range */
    if (globalTime > 1000.0f) globalTime -= 1000.0f;

    glutPostRedisplay();
}

/* Initialize OpenGL state */

```

```

void initGL(void)
{
    glClearColor(0.05f, 0.05f, 0.10f, 1.0f);
    initLighting();
}

/* Main */
int main(int argc, char **argv)
{
    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_DOUBLE | GLUT_RGB | GLUT_DEPTH);

    /* 1024 x 768 window */
    glutInitWindowSize(WIN_WIDTH, WIN_HEIGHT);
    glutInitWindowPosition(100, 50);
    glutCreateWindow("3 Bouncing Balls - Full Height, Never Stop");

    initGL();
    initBalls();

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutIdleFunc(idle);

    glutMainLoop();
    return 0;
}

```

Explanation of “why your balls are stopping” and how this fixes it:

- With gravity + bounce factor < 1, each bounce loses energy, so the motion decays and eventually looks like it stops.
- Here, I removed physical damping and instead use a periodic function:

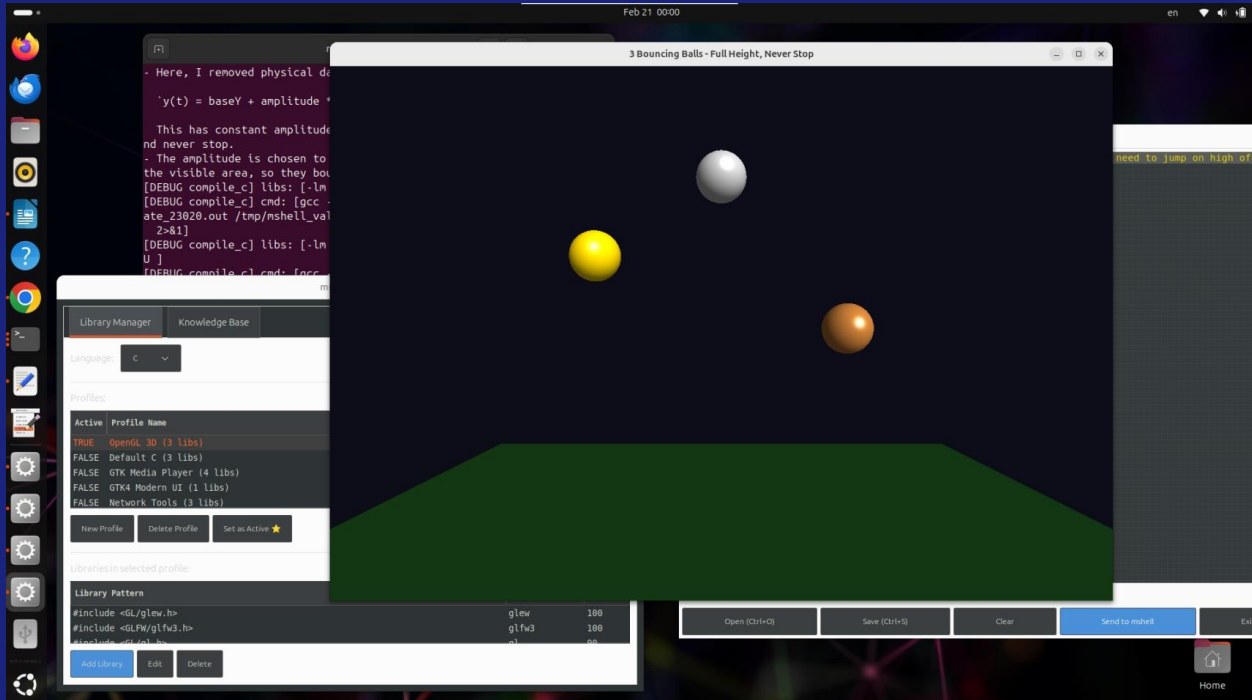
$$y(t) = \text{baseY} + \text{amplitude} * |\sin(\text{speed} * t + \text{phase})|$$
This has constant amplitude forever, so the balls never lose height and never stop.
- The amplitude is chosen to almost span from the bottom to the top of the visible area, so they bounce across nearly the full screen height.

[DEBUG compile_c] libs: [-lm -DFREEGLUT_STATIC -lglut]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_23020.out /tmp/mshell_validate_23020.c -lm -DFREEGLUT_STATIC -lglut 2>&1]

[DEBUG compile_c] libs: [-lm -DFREEGLUT_STATIC -lglut -lGL -IGLU -IGLU]

[DEBUG compile_c] cmd: [gcc -Wall -Wextra -std=c11 -o /tmp/mshell_validate_23020.out /tmp/mshell_validate_23020.c -lm -DFREEGLUT_STATIC -lglut -lGL -lGLU -lGLU 2>&1]



02/21/2026 Contacts: Igor Lukyanov (igor.lukanov@appservgrid.com),