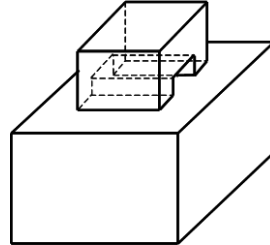
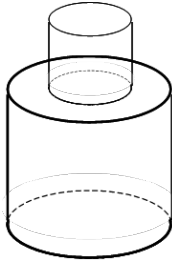


Lab07 for CPT205 Computer Graphics

Part 1. Exercise on Representations of 3D Models

1) Given the following objects below

- a) Identify if they are manifold or nonmanifold.
- b) Show if they obey the Euler's law.



2) A simple polyhedron is cut by a plane into two. Show that Euler's law holds for the new objects.

3) Investigate the main strengths and drawbacks of B-Rep model in terms of

- a) data available and data structure
- b) ease of modelling
- c) implementation requirements
- d) applications

4) Compare and contrast between wireframe, surface, CSG and B-Rep models, and further investigate their applications.

Part 2. Sample code for Boolean operations / geometric modelling

Compile and run the following code and observe how geometric modelling and Boolean operations work. Note that you can right click the mouse to see a simple menu for the operations. There are functions that will be explained and used in later lectures (e.g. hidden surface removal).

```
#define FREEGLUT_STATIC
#define GLUT_DISABLE_ATEXIT_HACK
#include <GL/freeglut.h>
#include <math.h>
#include <stdlib.h>

//Hide the terminal window
#pragma comment(linker, "/subsystem:\"windows\" /entry:\"mainCRTStartup\"")

#define SPHERE 1
#define CONE 2
#define CUBE 3

// csgOperation
// types of CSG operations
typedef enum {
    CSG_A,
    CSG_B,
    CSG_A_OR_B,
    CSG_A_AND_B,
    CSG_A_SUB_B,
    CSG_B_SUB_A
} csgOperation;

// globals
GLint Width;
GLint Height;

GLfloat zoom = 0.0;

GLfloat cone_x = 0.0;
GLfloat cone_y = 0.0;
GLfloat cone_z = 0.0;

GLfloat cube_x = 0.0;
GLfloat cube_y = 0.0;
GLfloat cube_z = 0.0;

GLfloat sphere_x = 0.0;
GLfloat sphere_y = 0.0;
GLfloat sphere_z = 0.0;

// GLint mouse_state = -1;
// GLint mouse_button = -1;

csgOperation Op = CSG_A_OR_B;

/* Both A and B are function pointers.
These function pointers are defined for pointing to the various shape functions, i.e. sphere(), cube()
and cone(). */
void(*A)(void);
void(*B)(void);

// functions

// one()
// draw a single object
void one(void(*a)(void)) // pointer as parameter
{
    glEnable(GL_DEPTH_TEST);
    a();
    glDisable(GL_DEPTH_TEST);
}

// or()
// boolean A or B (draw wherever A or B)
// algorithm: simple, just draw both with depth test enabled
void Or(void(*a)(void), void(*b)())
{

```

```

        // glPushAttrib(GL_ALL_ATTRIB_BITS); // TODO - should just push depth
        glEnable(GL_DEPTH_TEST);
        a(); b();
        // glPopAttrib();
    }

    // inside()
    // sets stencil buffer to show the part of A
    // (front or back face according to 'face')
    // that is inside of B.
    void inside(void(*a)(void), void(*b)(void), GLenum face, GLenum test)
    {
        // GLint i;
        // draw A into depth buffer, but not into color buffer
        glEnable(GL_DEPTH_TEST);
        glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
        glCullFace(face); //back is culled
        a();

        // use stencil buffer to find the parts of A that are inside of B
        // by first incrementing the stencil buffer wherever B's front faces
        // are

        glDepthMask(GL_FALSE);
        glEnable(GL_STENCIL_TEST);
        glStencilFunc(GL_ALWAYS, 0, 0);
        glStencilOp(GL_KEEP, GL_KEEP, GL_INCR);
        glCullFace(GL_BACK);
        b();

        // then decrement the stencil buffer wherever B's back faces are
        glStencilOp(GL_KEEP, GL_KEEP, GL_DECR);
        glCullFace(GL_FRONT);
        b();

        // now draw the part of A that is inside of B
        glDepthMask(GL_TRUE);
        glColorMask(GL_TRUE, GL_TRUE, GL_TRUE, GL_TRUE);
        glStencilFunc(test, 0, 1);
        glDisable(GL_DEPTH_TEST);
        glCullFace(face);
        a();

        // reset stencil test
        glDisable(GL_STENCIL_TEST);
    }

    // fixup()
    // fixes up the depth buffer with A's depth values
    void fixup(void(*a)(void))
    {
        // fix up the depth buffer
        glColorMask(GL_FALSE, GL_FALSE, GL_FALSE, GL_FALSE);
        glEnable(GL_DEPTH_TEST);
        glDisable(GL_STENCIL_TEST);
        glDepthFunc(GL_ALWAYS);
        a();

        // reset depth func
        glDepthFunc(GL_LESS);
    }

    // and()
    // Boolean A and B (draw wherever A intersects B)
    // algorithm: find where A is inside B, and then find where
    //             B is inside A
    void And(void(*a)(void), void(*b)(void))
    {
        inside(a, b, GL_BACK, GL_NOTEQUAL);
#ifdef 1 // set to 0 for faster, but incorrect results
        fixup(b);
#endif
        inside(b, a, GL_BACK, GL_NOTEQUAL);
    }

    // sub()
    // Boolean A subtract B (draw wherever A is and B is NOT)

```

```

// algorithm: find where a is inside B, and then find where
//             the BACK faces of B are NOT in A
void sub(void(*a)(void), void(*b)(void))
{
    inside(a, b, GL_FRONT, GL_NOTEQUAL);
#ifdef 1 // set to 0 for faster, but incorrect results
    fixup(b);
#endif
    inside(b, a, GL_BACK, GL_EQUAL);
}

// sphere()
// draw a white sphere
void sphere(void)
{
    // glLoadName(7);
    glPushMatrix();
    glTranslatef(sphere_x, sphere_y, sphere_z);
    glTranslatef(4, 0, 0);
    glColor3f(1.0, 1.0, 1.0);
    glutSolidSphere(5.0, 16, 16);
    glPopMatrix();
}

// cube()
// draw a red cube
void cube(void)
{
    // glLoadName(4);
    glPushMatrix();
    glTranslatef(cube_x, cube_y, cube_z);
    glColor3f(1.0, 0.0, 0.0);
    glutSolidCube(8.0);
    glPopMatrix();
}

// cone()
// draw a green cone
void cone(void)
{
    // glLoadName(9);
    glPushMatrix();
    glTranslatef(cone_x, cone_y, cone_z);
    glColor3f(0.0, 1.0, 0.0);
    glTranslatef(0.0, 0.0, -6.5);
    glutSolidCone(4.0, 15.0, 16, 16);
    // glRotatef(180.0, 1.0, 0.0, 0.0);
    // glutSolidCone(4.0, 0.0, 16, 1); // cone of 0 height
    glPopMatrix();
}

void init(void)
{
    GLfloat lightposition[] = { -3.0, 3.0, 3.0, 0.0 };

    glDepthFunc(GL_LESS);
    glEnable(GL_DEPTH_TEST);

    glEnable(GL_LIGHT0);
    glEnable(GL_LIGHTING);
    glLightfv(GL_LIGHT0, GL_POSITION, lightposition);
    glLightModeli(GL_LIGHT_MODEL_TWO_SIDE, GL_TRUE);

    glEnable(GL_COLOR_MATERIAL);

    glEnable(GL_CULL_FACE);

    glClearColor(0.0, 0.0, 1.0, 0.0);
}

void reshape(int width, int height)
{
    Width = width;
    Height = height;

    glViewport(0, 0, width, height);
}

```

```

    glMatrixMode(GL_PROJECTION);
    glLoadIdentity();
    glFrustum(-3.0, 3.0, -3.0, 3.0, 64, 256);

    glMatrixMode(GL_MODELVIEW);
    glLoadIdentity();
    glTranslatef(0.0, 0.0, -200.0 + zoom);
}

void display(void)
{
    glClear(GL_COLOR_BUFFER_BIT | GL_DEPTH_BUFFER_BIT | GL_STENCIL_BUFFER_BIT);

    // glPushMatrix();
    glPushMatrix();
    glRotatef(5, 1, 0, 0);
    glRotatef(5, 0, 1, 0);
    switch (Op) {
    case CSG_A:
        one(A);
        break;
    case CSG_B:
        one(B);
        break;
    case CSG_A_OR_B:
        Or(A, B);
        break;
    case CSG_A_AND_B:
        And(A, B);
        break;
    case CSG_A_SUB_B:
        sub(A, B);
        break;
    case CSG_B_SUB_A:
        sub(B, A);
        break;
    }
    glPopMatrix();
    // glPopMatrix();
    glFlush();
}

void keyboard(unsigned char key, int x, int y)
{
    switch (key) {
    case 'c':
        if (A == cube && B == sphere) {
            A = sphere;
            B = cone;
        }
        else if (A == sphere && B == cone) {
            A = cone;
            B = cube;
        }
        else { // if(A == cone && B == cube)
            A = cube;
            B = sphere;
        }
        break;
    case 'a':
        Op = CSG_A;
        break;
    case 'b':
        Op = CSG_B;
        break;
    case '|':
        Op = CSG_A_OR_B;
        break;
    case '&':
        Op = CSG_A_AND_B;
        break;
    case '-':
        Op = CSG_A_SUB_B;
        break;
    case '_':
        Op = CSG_B_SUB_A;
        break;
    }
}

```

```

        case 'z':
            zoom -= 6.0;
            reshape(Width, Height);
            break;
        case 'Z':
            zoom += 6.0;
            reshape(Width, Height);
            break;
        case 27:
            exit(0);
            break;
        case '\r':
            break;
        default:
            return;
    }

    glutPostRedisplay();
}

void menu(int item) //mouse and keyboard can control the display altogether
{
    keyboard((unsigned char)item, 0, 0);
}

int main(int argc, char** argv)
{
    int ops, zoom;

    glutInit(&argc, argv);
    glutInitDisplayMode(GLUT_RGB | /*GLUT_DOUBLE */ GLUT_DEPTH | GLUT_STENCIL);
    glutInitWindowPosition(100, 100);
    glutInitWindowSize(500, 500);
    glutCreateWindow("CSG Operations Demo");

    glutDisplayFunc(display);
    glutReshapeFunc(reshape);
    glutKeyboardFunc(keyboard);
    // glutMouseFunc(mouse);
    // glutMotionFunc(motion);

    ops = glutCreateMenu(menu);
    glutAddMenuEntry("A only      (a)", 'a');
    glutAddMenuEntry("B only      (b)", 'b');
    glutAddMenuEntry("A or B      (|)", '|');
    glutAddMenuEntry("A and B     (&)", '&');
    glutAddMenuEntry("A sub B     (-)", '-');
    glutAddMenuEntry("B sub A     (_)", '_');
    zoom = glutCreateMenu(menu);
    glutAddMenuEntry("Zoom decrease (z)", 'z');
    glutAddMenuEntry("Zoom increase (Z)", 'Z');
    glutCreateMenu(menu);
    glutAddMenuEntry("CSG Operations Demo", '\0');
    glutAddMenuEntry("      ", '\0');
    glutAddSubMenu("Operations", ops);
    glutAddSubMenu("Zoom", zoom);
    glutAddMenuEntry("      ", '\0');
    glutAddMenuEntry("Change shapes (c)", 'c');
    glutAddMenuEntry("      ", '\0');
    glutAddMenuEntry("Quit (Esc)", '\033');
    glutAttachMenu(GLUT_RIGHT_BUTTON);

    init();

    A = cube;
    B = sphere;

    glutMainLoop();
}

```

