

Practical 2 **Constructive Solid Geometry**

1 Introduction

The purpose of this practical is to help you get some hands-on experience of CSG (constructive solid geometry). You will manipulate some 3D graphical shapes and think about the algebraic operations involved.

You are encouraged to use sVLIs to complete this practical. You may need to be physically in the lab to do this because running sVLIs remotely is quite a slow process. An alternative to using the lab machines (physically or remotely) is to run it on your own machine, but this will involve building sVLIs, the code for which is fairly outdated. I will do my best to keep my fork of the code at <https://github.com/jmpf/SvLis> as up-to-date as possible.

You have the freedom to use any other geometric modeller which offers Boolean operations. You have a choice between using primitives and set theory operations in either

- **blender** (which is not a true CSG modeller but has the right operations) or
- **openscad** (which is fully CSG).

You will need to look up instructions on-line.

Regardless of your modeller of choice you may earn an S grade by also modelling a mug. You can earn an S+ by modelling something which is demonstrably more creative!

2 sVLIs information

The bulk of the instructions for this practical continue to detail the steps you need to follow in order to construct the mug using the sVLIs geometric modeller.

If you wish to use it remotely, you will need to set up X11 forwarding, which will display on your local machine a graphical window from the linux window manager on the departmental network. Depending on your local operating system, you may need extra `ssh` or `X11` software. Then log in with: `ssh -X username@ecs.ox.ac.uk` or similar.

3 Tasks

If you decide to use an alternative modeller, adapt these tasks to fit your environment.

1. Commands you need with **svLIs**

In order to get the necessary files either download the zip file from the website or copy everything from the appropriate directory:

```
$ cp /usr/local/practicals/gmod/SolidGeometry/* .
```

You will be modifying the C++ program called `CSGmug.cxx`. To compile your code, you can use the command

```
$ make CSGmug
```

and to run it, use

```
$ ./CSGmug
```

This will cause a graphics window to appear, and in your terminal the computer politely requests ‘`type any character to finish:`’. If, at any time, you want to close the graphics window and quit the program, you need to type into your terminal window any character followed by `Enter`.

2. What you can do with the graphics

Once you have a graphics window displaying a geometric model, you can turn the model around with your left mouse button, or shift it along with the middle mouse button.

Keep your mouse over the window and type ‘?’ . This will print (in your terminal window) a list of other commands which you can type in the graphics window. Try a few, such as: `c`, `w`, `f`, `F`, `h`, `0` (zero). Make a note of each as they will be useful later.

3. Shapes available

Now look at the program in the editor window, and focus on this line:

```
result = a_cuboid;
```

This defines the result that gets displayed in the graphics screen. If you look at the lines above it, you will discover that `c` was defined somewhere as a `cuboid`.

As you can see, there are other shapes available.

4. Putting together a geometric model

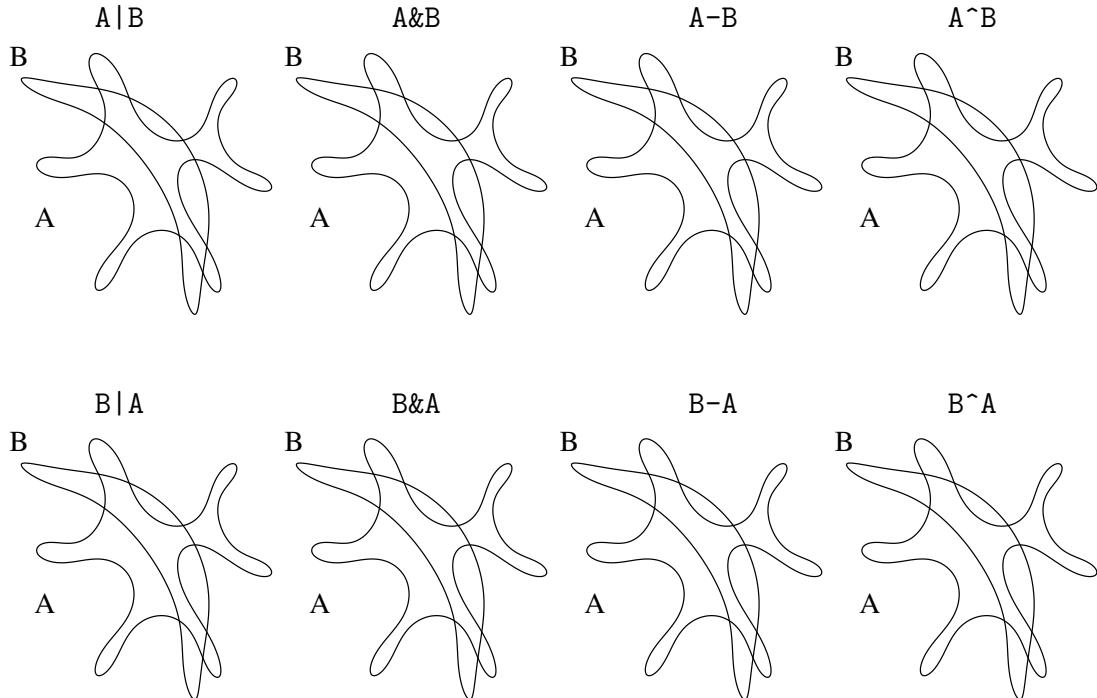
Now, by selectively uncommenting some of the lines of code, try combining the shapes available with some of the operators `|`, `&`, `-`, `^`

```
result = a_cuboid | a_sphere;    or    result = a_cuboid & a_sphere;    or  
result = a_cuboid - a_sphere;    or    result = a_cuboid ^ a_sphere;
```

What shapes do you get?

5. Identifying useful operations (no need to show this in the practical)

Based on experiments that you have made with the program, can you now illustrate what these operations will produce? Given the two 2D figures A and B in the diagrams below, colour in the results of each operation.



6. Have the manual handy

The sVLIs manual is available on the teaching network at
file:///usr/local/practicals/gmod/SvlisDocumentation/svlis.html

Pay particular attention to the `sv_line` definition, which expects a direction vector `dir` and a point `org`, so that the equation of the infinite line is `org + λ dir`:

```
sv_line l = sv_line(sv_point dir, sv_point org); //Non-intuitive order
```

7. Ray trace a model

The interactive window and some ray tracing functionality can be toggled on or off. The default is to have the interactive window on, but the ray tracing off. The program always makes a `.mod` file.

Find the code:

```
bool plot = true;
bool raytrace = false;
if (plot) {
    ...
} else if (raytrace){
    // Raytrace to file "raytraced.ppm"
    sv_integer width=200; //pixels. 800 is good
    sv_integer height=150; //pixels. 600 is good
    ...
}
```

Switch the `plot` and `raytrace` variables to `false` and `true` respectively. Compile and run your code again. Experiment with different window sizes or viewing parameters.

8. The svLIs test harness

svLIs comes with a test harness (as demonstrated in the lectures). The idea of this program is to provide a rapid way to combine basic primitive shapes without having to go through the edit, compile, run cycle.

Run the test harness program with

```
$ svlis
```

Follow the on screen instructions and try combining some shapes. Try to load the output of your program (`mug.mod`) into the test harness.

9. Make a mug (your main task)

There are functions to return simple shapes as sets. These are:

```
sv_set_cuboid(sv_point most_neg, sv_point most_pos)
```

```
sv_set_cuboid(sv_box b)
```

both generate the intersection of six planes needed to represent the cuboid required.

```
sv_set_cylinder(sv_line axis, sv_real radius)
```

returns a cylinder with given axis and radius.

```
sv_set_cone(sv_line axis, sv_real angle) // angle in radians
```

returns a cone. Or rather half of an infinite cone: the half that points the same way as the line.

```
sv_set_sphere(sv_point centre, sv_real radius)
```

returns a sphere with given centre and radius.

```
sv_set_torus(sv_line l, sv_real big_r, sv_real little_r)
```

returns a torus the major circle of which is perpendicular to the line, and which has its centre at the line's origin. The variable `little_r` is the torus's minor radius.

```
sv_set_cyclide(sv_line l, sv_point sym, sv_real big_r, sv_real little_r,  
sv_real rc)
```

is a cyclide with its major circle (`radius big_r`) perpendicular to the line and which has its centre at its origin. `little_r + rc` is the biggest minor radius of the cyclide; `little_r - rc` is the smallest minor radius of the cyclide. The cyclide is symmetrical about the direction `sym`, which obviously must not be parallel to the direction of axis.

Now make a mug that looks like this



Hint: to avoid creating infinitely-thin planes, do not use components that ‘just touch’. Overlap them instead.

4 Optional

Model another object, combining at least three primitives.

Email me an image that I can put on the course website.

*Joe Pitt-Francis
Joe.Pitt-Francis@cs.ox.ac.uk*

The software package called sVLIs is a CSG modeller and was developed in the 1990s by a group of researchers at the University of Bath.