Creating a 2D CSG software

Introduction

Constructive Solid Geometry (CSG) is a way to model 3D objects based on shapes and logical operators. It is widely used in industry to model parts like engines, cogs, and so on. In this project, your group will have to implement a software that does some CSG operations in 2D. Your client (i.e., the lecturer) would like a java or C++ implementation of 2D CSG operators that makes use of Object-Oriented programming concepts. Some of the requirements for the software are informally given below, but you will have to express them in a formal way for your project and report. We do expect most user requirements to be implemented, though some of them may require time and more advanced technical skills. Overall, a basic implementation can still be OK if the project methodology (i.e., the software engineering aspects) used to create software is good. Again, an important aspect of the project is the use of an Object-Oriented language that should simplify the programming of your application but at the same time demonstrate your understanding of this programming paradigm.

We recommend that you get familiar with some CSG concepts by having a look at two open projects first: OpenSCAD (http://www.openscad.org/) and OpenJSCAD (http://openjscad.org/).

Informal requirements from the client

1) Project is implemented in Java or C++ and makes much use of the Object-Oriented possibilities those languages offer (like inheritance and templates/generic types). Have a discussion with your lecturer if you are thinking about using another language.
2) The CSG model makes use of various classes for the internal representation of a shape. For instance:
    - Tree: Internally, a tree is a (possibly binary) tree of nodes.
    - Nodes shall be either: Shapes, Transforms, Mix, a reference to another node (difficult, no cycles allowed) or For loops (difficult).
    - Shapes can be either:
        - Poly(nb_of_faces, Diameter=1)
            - Triangle(Diameter=1)
            - Square(Diameter=1)
        - Circle(Diameter=1);
        - Closed curves (optional).
    - Transforms:
        - Rotate(angle)
        - Scale(zoom)
        - Translate(x,y)
        - For(n, transform, node)
    - Mix:
        - +,-,& **or** Union, Diff, Inter

3) The CSG model will be represented internally as a tree of objects, but a user interface should allow Interacting with a text representation of the tree and displaying a graphical version of the 2D model as well. The exact specifications of the tree structure are left to the contractor, but

they should be logical, conflict/ambiguity-free and well-specified in the document. An Abstract Data Type (ADT) for the tree is wished as part of the more formal specifications.

4) The tree should be saveable and loadable from a file.
5) Nodes should have optional display names and variable names as members of a class.
6) Constructors should be widely available and allow creating trees. For instance, one could create models in the following ways:
    a. Node t1= Triangle(3); //C++
    b. Node t2= Union(Triangle(), Square()); //C++, or as
    c. Node t3=Triangle(1.5)+Square(); // C++, with operator+ overloading.
    d. Node t4= Rotate(PI/2., Intersection(Triangle(),Square()));//C++

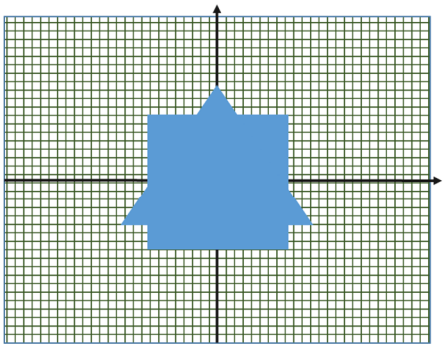

Fig 1: A 2D CSG graphics of t3

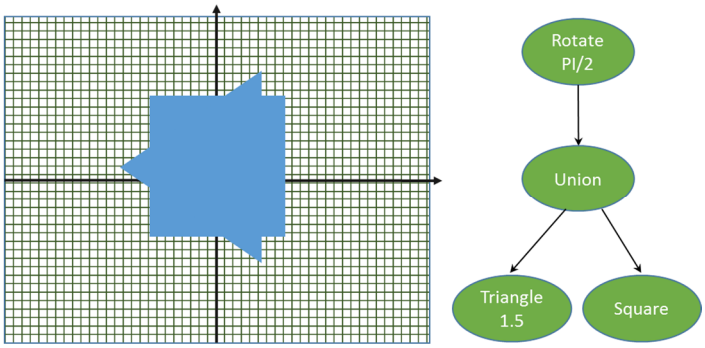

Fig 2: A 2D CSG graphics of t4

Suggestive marking scheme for the implementation:

- First class: A graphical display of the 2D graph making sure small trees can be displayed in a reasonable amount of time. Various optional features have been added to the application and to the CSG language.

- Merits: The possibility to have a text window to create or display a tree. Object-oriented concepts well applied. Project is easily extensible.

- Pass: Basic class hierarchy implemented, with some functionalities available like the possibility to load or save a tree.

CSC M94 project - Group organisation

You will be meeting weekly with your group as your project shall be a continuous collaborative effort during the term. One of the group meetings will happen during the regular lecture time slot, but you should also have at least one extra meeting per week. All meetings should have minutes that shall include: Who is present, who is not, progress on the report and on the software aspects, outstanding actions to be completed, what need to be done next (actions), status of project, steps to be undertaken, who is doing what and anything else that is relevant to the project.

A format should be agreed from the start for the minutes. A collection of weekly minutes will be sent weekly as a pdf file to the lecturer with the subject "CSCM94 Weekly minutes group X".

Suggested Schedule

- Week 2: Knowing your group partners. Description of everyone's role. Who is doing what? What kind of tools to be used (coding, code management, repository). How to document. Final document preparation. Format of documents like minutes.
- Week 3: Plans for the project. Quality assurance. Life cycle, documentation in place, etc… Tools to be used, etc… : List of specifications for the project established inside the document. Work distribution.
- Week 4: Preliminary design: Class diagram and possibly draft of a user interface (state diagram).
    - **A preliminary but complete class diagram must be emailed to the lecturer by end of week 4.**
- Week 5: Revised design/coding.
- Week 6: unit coding.
- Week 7: unit coding.
- Week 8: Coding.

- Week 9: Testing .
- Week 10: 1.0 version is out + demo + project document with code.

You can be ahead of schedule, but you do not really want to be late.

Final Document

The final document that you will submit in week 10 (or before if you wish to) is expected to reflect the work described in the minutes and should include for instance:

- Abstract.
- Introduction to document.
- Description of the company.
- Quality section on tools, schedule.
- A formal list of User Requirements.

- Preliminary design including a class diagram, details of functions to be coded, class members, etc…
- Test sections.
- Result Section (informally written, screenshots, etc…
- Narrative conclusion reflecting on the overall project.
- Appendix
    - Minutes.
    - Code.
    - **Contribution report.**

Important note on how you will document individual efforts in the team (contribution report) !

As group projects are always subject to discussions about the contribution of each member, we will ask you to do as follows:

- Each member will have to submit his own report, which should ideally be the same as what the other group members have submitted. We do not consider plagiarism cases between members of a common group, so you are free (and encouraged here) to submit similar reports and possibly get the same mark for them!

- An appendix on your individual effort is required. This will be considered for marking, especially if effort between team members is not shared. This should be written by yourself, though you may want to discuss it within the team, and it should contain:
    - A subjective quantitative evaluation of the **percentage** of **your** contribution to the project and/or assignments. (e.g., ideally 25% if there are 4 team members).

- A qualitative statement on what **you** did exactly. You may want to keep it short, but this is just a recommendation and you will have to make sure that your contribution to the project is clear and at the same time it does not contradict what other group members have said in their reports.

The report mark should be closely related to evidence that you have employed a good software engineering methodology. You should make sure that the Software Engineering sections are written in a formal and concise way. Code and demo will also be important elements to get an idea of how successful your project was.

CSC M94 Group members

- Group 1
  - 119332
  - 847620
  - 689376
  - 804658

- Group 2
  - 689463
  - 710739
  - 360437
  - 843927

- Group 3
  - 849119
  - 263529
  - 689253
  - 688283

- Group 4
  - 404737
  - 689092
  - 689404
  - 882230

- Group 5
  - 845632
  - 848536
  - 843363
  - 845231

- Group 6
  - 847800
  - 845597
  - 843356