Improved CSG Rendering using Overlap Graph Subtraction Sequences

Nigel Stewart, Geoff Leach
RMIT School of Computer Science
and Information Technology
{nigels,gl}@cs.rmit.edu.au

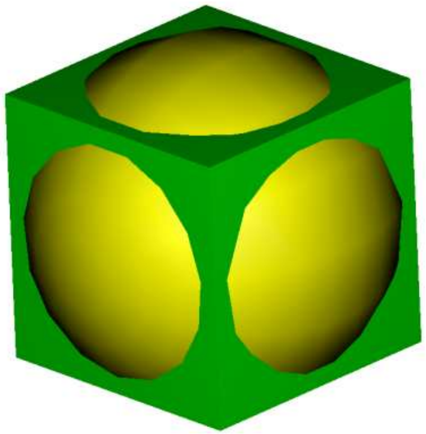
Sabu John
Department of Mechanical and Manufacturing Engineering
Sabu.John@rmit.edu.au

RMIT University, Melbourne, Australia

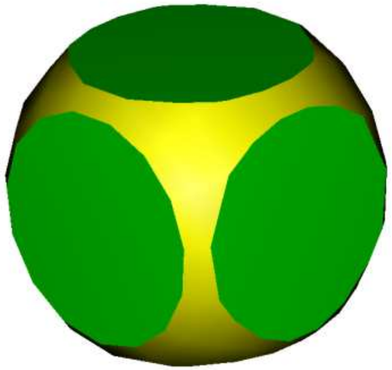
Overview

- CSG Rendering
- Sequenced Convex Subtraction (SCS)
- Permutation Embedding Sequences
- Graph-based Subtraction Sequences
- Experimental Results
- Conclusion, Demonstration

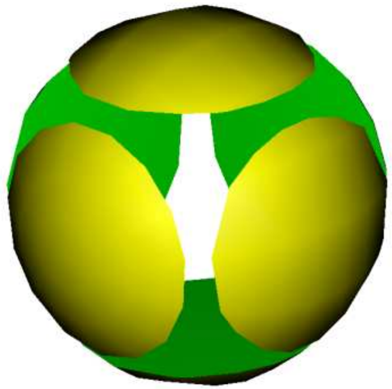
CSG Rendering



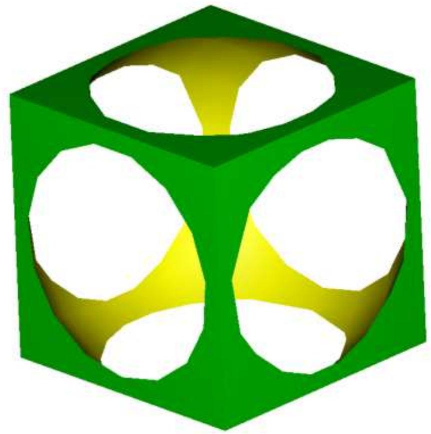
Union
$A \cup B$



Intersection
$A \cap B$

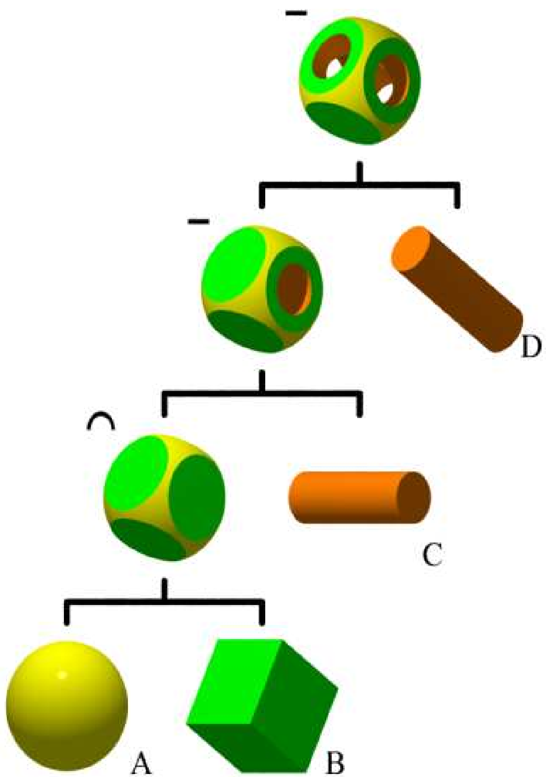


Difference
$A - B$



Difference
$B - A$

CSG Tree



Previous CSG Rendering Algorithms

Trickle Algorithm

*D. Epstein, F. Jansen, J. Rossignac, **Z-Buffer Rendering from CSG: The Trickle Algorithm**, IBM Research Report RC 15182, Nov 1989*

Goldfeather Algorithm

*J. Goldfeather, S. Molnar, G. Turk, H. Fuchs, **Near Real-Time CSG Rendering Using Tree Normalization and Geometric Pruning**, IEEE CG&A, Vol. 9, No. 3, May 1989, pp. 20-28*

Trickle Algorithm

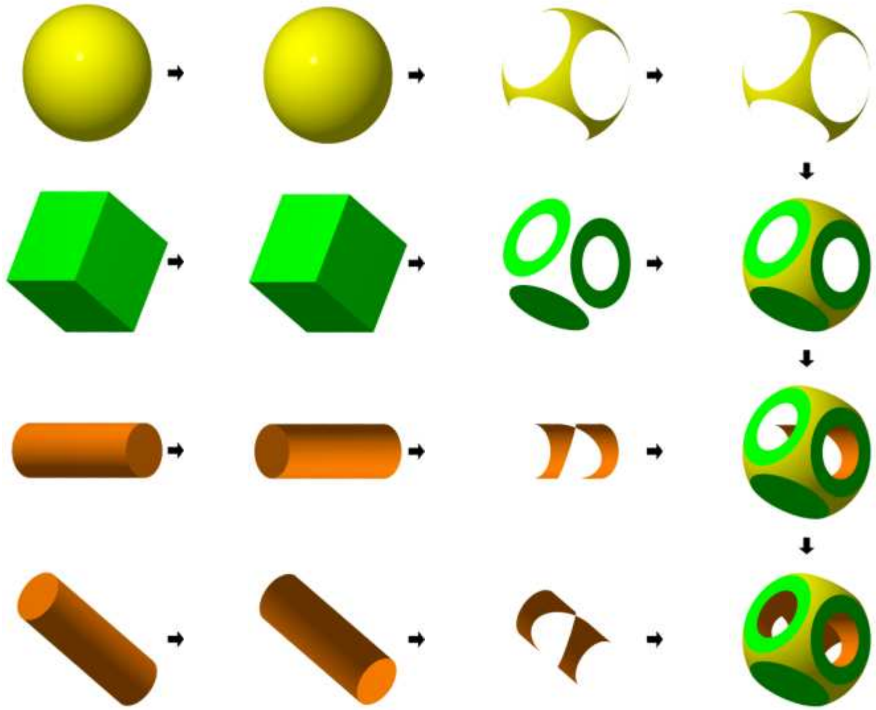
- ‘Trickle’ from front to back
- Subtract volumes represented as z-buffer pairs
- Implemented in OpenGL:
  - Requires multiple z-buffers, expensive to emulate
  - Requires multiple z-tests, expensive to emulate
- Could be revisited in light of graphics hardware improvements.

Goldfeather Algorithm

- CSG Tree normalisation step
- Clip shapes individually in the z-buffer
- Implemented in OpenGL
  - Requires $O(n^2)$ time
  - $O(n)$ z-buffer copy operations
- Enhancements
  - Depth complexity sampling results in $O(kn)$ time where $1 \leq k \leq n$
  - Improvements suggested by Tobler et. al. at VRVis

Goldfeather Algorithm

z_{output} is output depth buffer
z_{tmp} is temporary depth buffer
$z_{output} \leftarrow Z_{far}$
for all products P **do**
  for all objects $Q \in P$ **do**
    if Q subtracted **then**
      $z_{tmp} \leftarrow Q_{back}$
    if Q intersected **then**
      $z_{tmp} \leftarrow Q_{front}$
    for all objects $R \in P$ **do**
      if $R \neq Q$ **then**
        clip z_{tmp} against R
    for all pixels **do**
      if $z_{tmp} < z_{output}$ **then**
        $z_{output} \leftarrow z_{tmp}$



Motivation for SCS CSG Rendering Algorithm

- Object-space approaches well understood, commonly used
- Performance issues with Trickle and Goldfeather algorithms
- Real-time visualisation and editing of complex CSG shapes
- Rapid evolution of graphics hardware
- Efficient handling of large numbers of convex objects

SCS Algorithm

N. Stewart, G. Leach, S. John,
A CSG Rendering Algorithm for Convex Objects*,*
The 8-th International Conference in Central Europe on Computer Graphics, Visualisation and Interactive Digital Media '2000 - WSCG 2000
Volume II, pp. 369-372

N. Stewart, G. Leach, S. John
Linear-time CSG Rendering of Intersected Convex Objects*,*
The 10-th International Conference in Central Europe on Computer Graphics Visualization and Computer Vision '2002 - WSCG 2002
Volume II, pp. 437-444

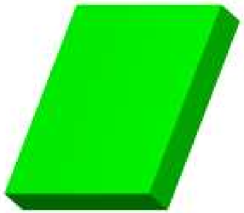
SCS Algorithm

- Sequenced Convex Subtraction (SCS)
  - Make use of Trickle ‘front-to-back’ subtraction approach
  - Make use of Goldfeather tree normalisation
  - Convex shapes

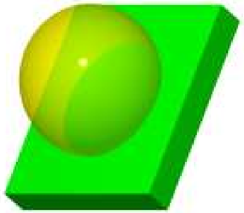
- Contributions:
  - Intersection of convex objects without z-buffer copying
  - Subtraction of convex objects without z-buffer copying
  - Permutation embedding sequences

SCS Subtraction

Subtract one convex shape from the z-buffer

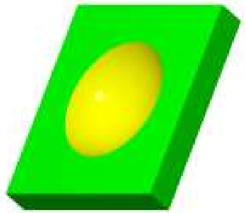


(a)
Draw initial surface into z-buffer



(b)
Compare z-buffer to front and back surfaces

(c)
Flag pixels inside volume

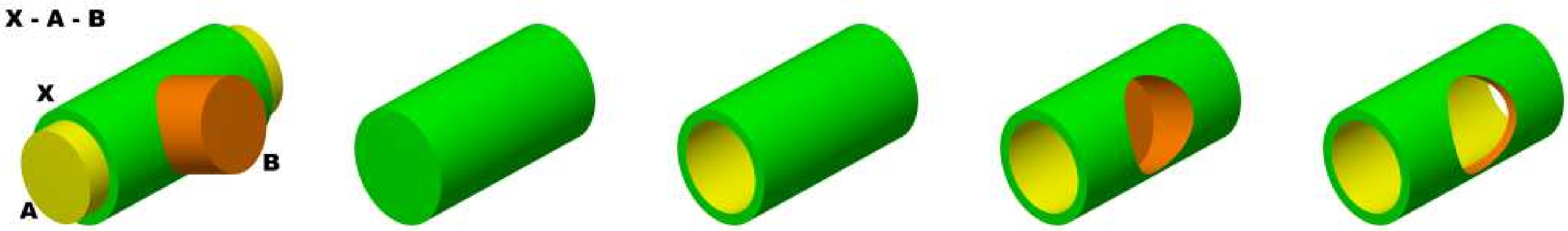


(d)
Replace flagged pixels with back surface

(e)
Reset regions of complete subtraction

Sequenced Subtraction

- The z-buffer can store only one surface per pixel.
- Each shape may need to be subtracted more than once.
- If possible, subtract in front-to-back order



CSG Tree | X | X − A | X − A − B | X − A − B − A

Permutation Embedding Sequences

- Not always possible to subtract from front to back
  - Front-to-back order not always known
  - Necessary sequence may vary from pixel to pixel
  - Use a sequence that 'embeds' all possible sequences
- Permutation embedding sequences embed all $n!$ permtuations of n objects.
  - The sequence aba embeds ab and ba
  - The sequence $abacaba$ embeds abc, acb, bac, bca, cab and cba.
- What is the shortest sequence embedding all permutations of n objects?

Permutation Embedding Property

Sequences		Combined Sequence		Embedded Sequences
abc	$\longrightarrow$	$abacaba$	$\longrightarrow$	$ab\star c\star\star\star$
acb				$a\star\star c\star b\star$
bac				$\star bac\star\star\star$
bca				$\star b\star ca\star\star$
cab				$\star\star\star cab\star$
cba				$\star\star\star c\star ba$

Subtraction Sequences for CSG Rendering

- Permutation embedding sequences are $O(n^2)$ or $O(kn)$ in length

- Sample subtraction sequences:
  $n = 4 \rightarrow abcdabcadbac$ (length 12)
  $n = 5 \rightarrow abcdeabcdaebcadbcea$ (length 19)

- Optimal length permutation embedding sequences are not known for $n > 5$

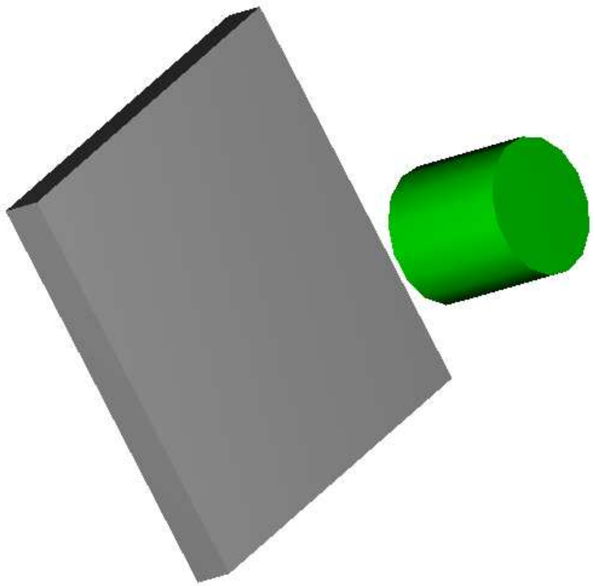
Overlap Graph Subtraction Sequences

- Use object-space spatial overlap of convex 3D shapes
  - Bounding box test
  - Constrain subtraction sequence to necessary dependencies

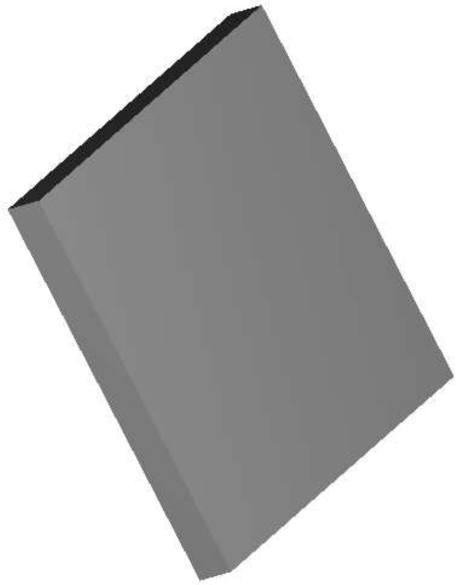
- Make use of an *Overlap-Graph*
  - Each node is a shape in the CSG tree
  - Each edge corresponds to spatial overlap between pairs of shapes
  - Subtraction sequence need only embed each acyclic path

External Subtracted Objects

Subtracted objects not touching all intersected objects can be ignored



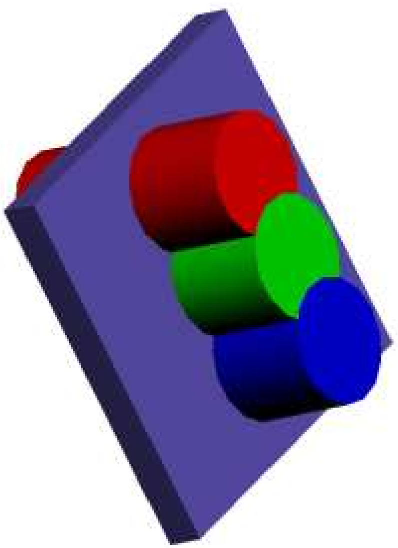
Subtracted Cylinder



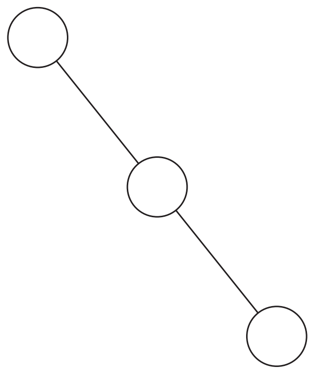
Rendered Result

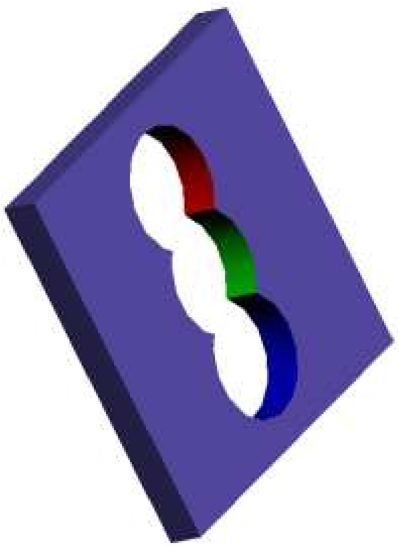
Acyclic Node Trimmming

Two possible acyclic paths: abc and cba, others need not be embedded.
Subtraction sequence $abcba$ may be used, rather than $abacaba$.



Subtracted Cylinders

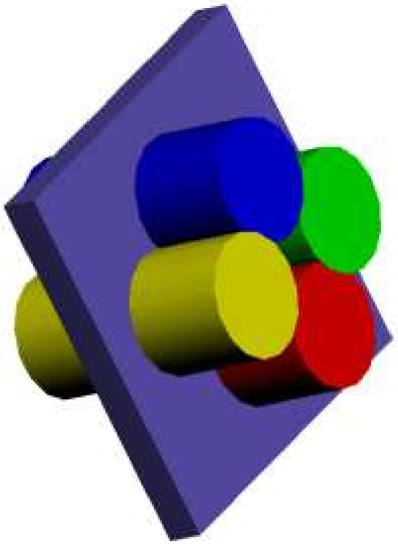




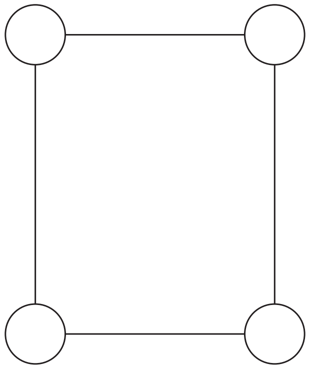
Rendered Result

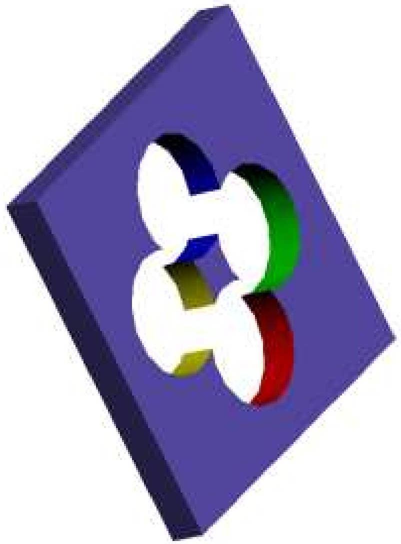
‘Loop’ Overlap Graph

Only clockwise and anti-clockwise paths need to be embedded.
Subtraction sequence for loops are $O(n)$ in length.



Subtracted Cylinders

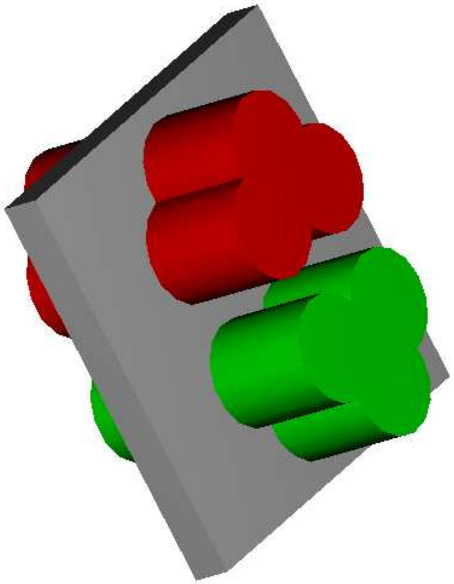




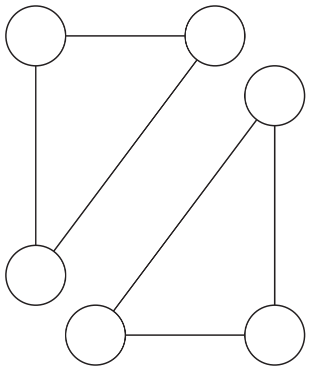
Rendered Result

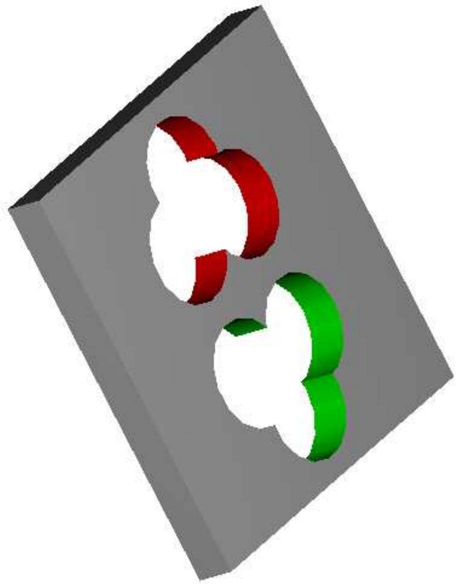
Disconnected Graphs

Each connected subgraph is encoded separately.
There can be no sequence of subtraction between graphs.



Subtracted Cylinders

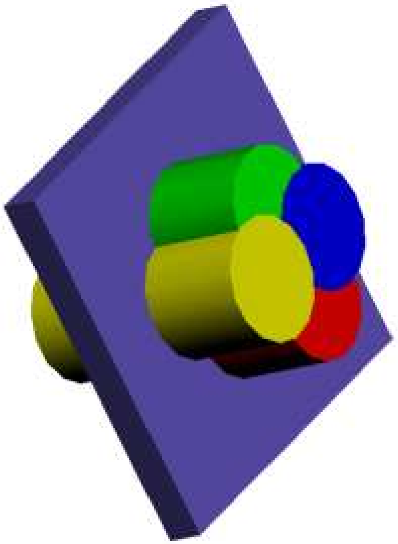




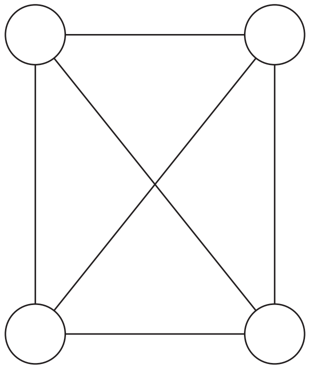
Rendered Result

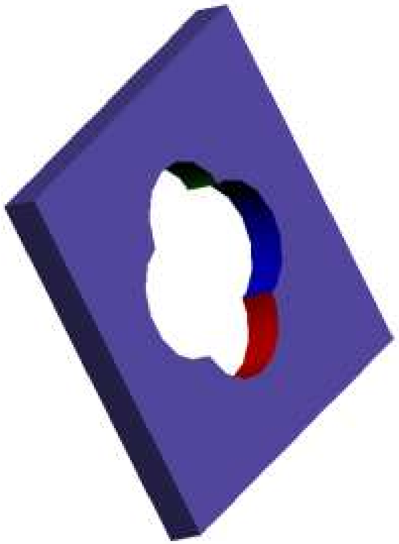
Cyclic Overlap Graph

All permutations need to be embedded.
Subtraction sequence for loops are $O(n^2)$ or $O(kn)$ in length.



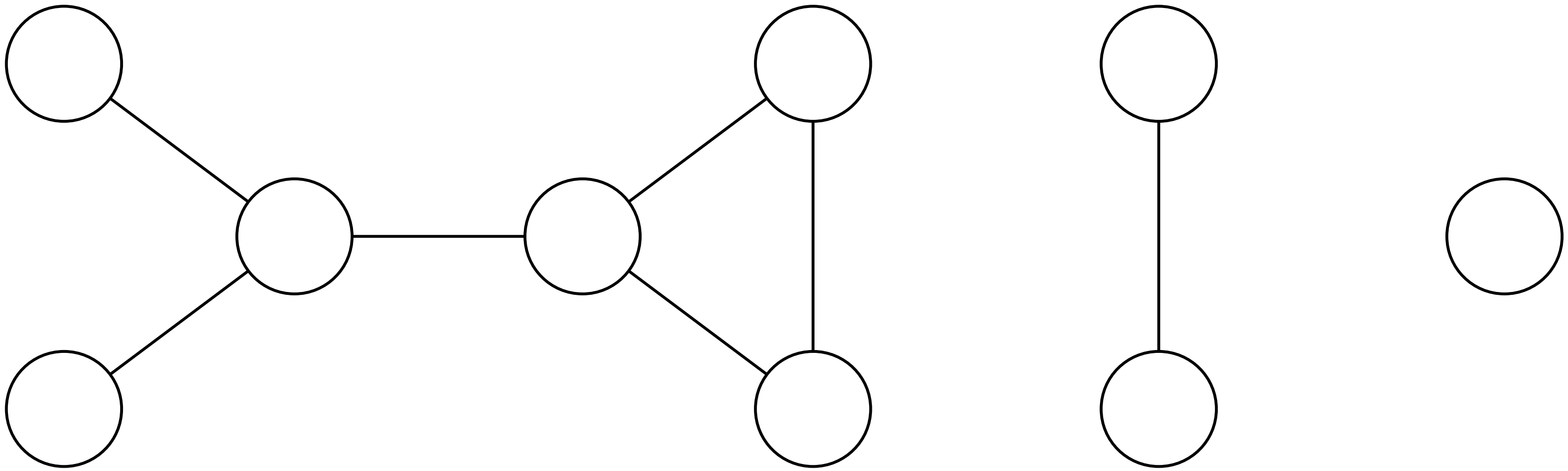
Subtracted Cylinders



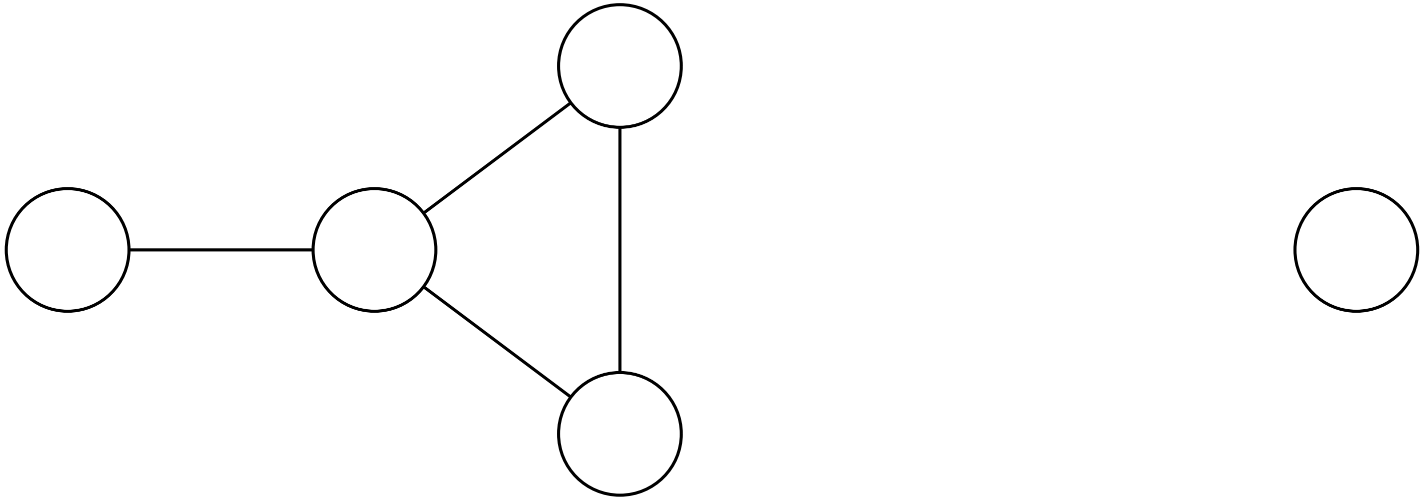


Rendered Result

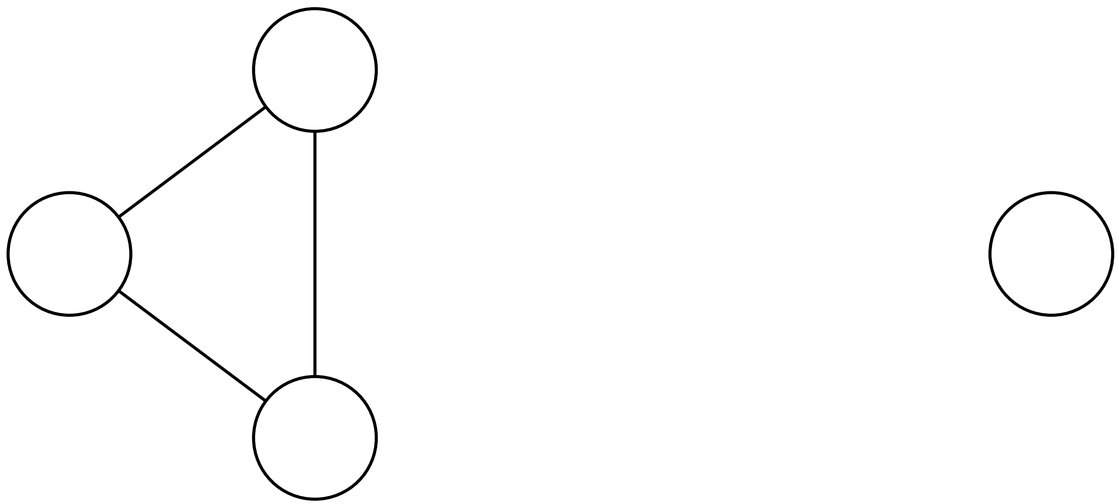
Example (i)



Example (ii)

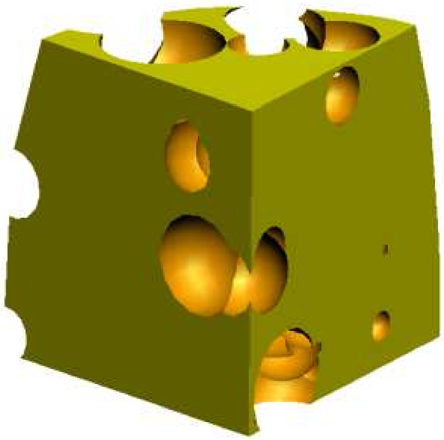


Example (iii)

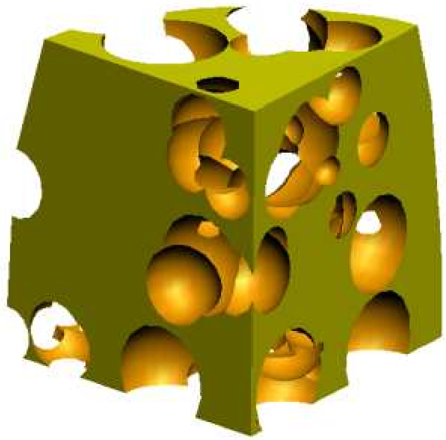


Swiss Cheese Experiment

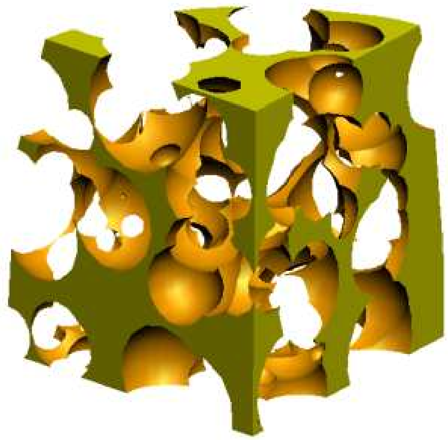
Spherical holes positioned and scaled randomly.



50 Subtracted Holes

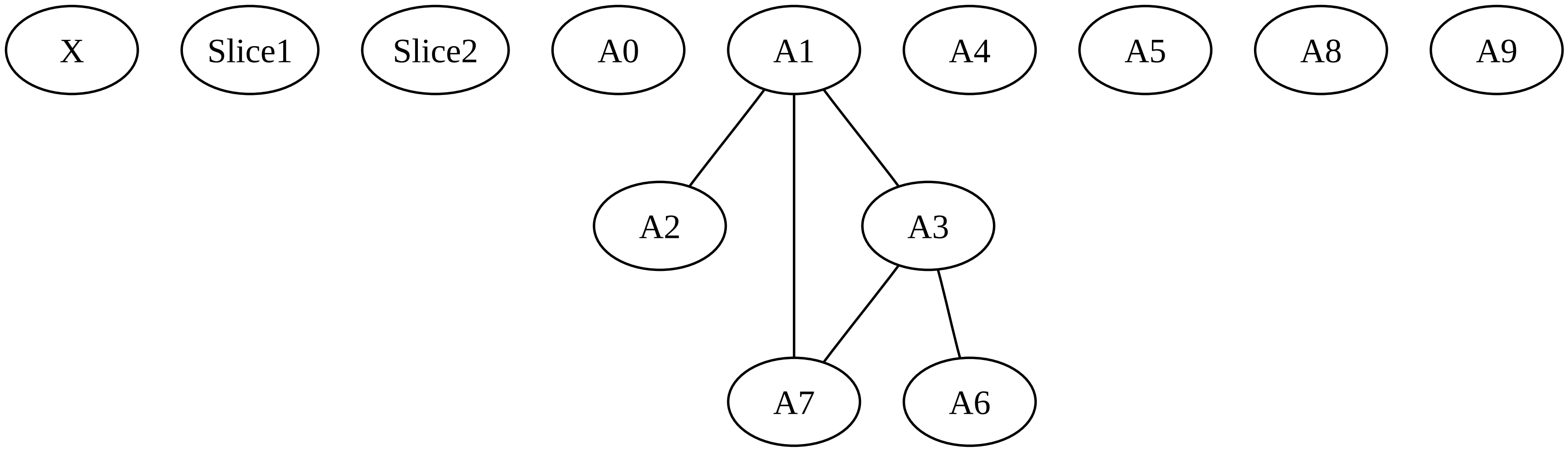


100 Subtracted Holes

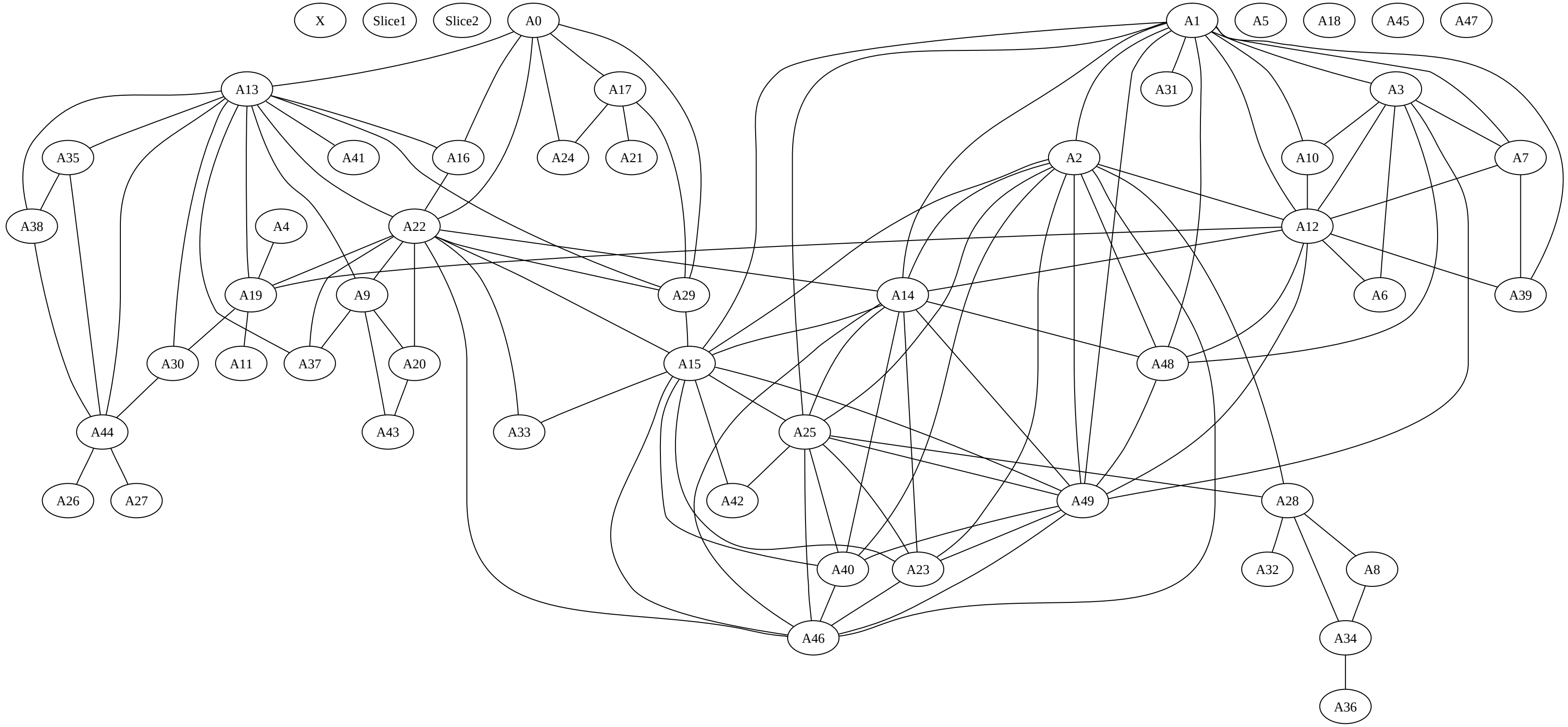


200 Subtracted Holes

Swiss Cheese Overlap Graph ($n = 10$)



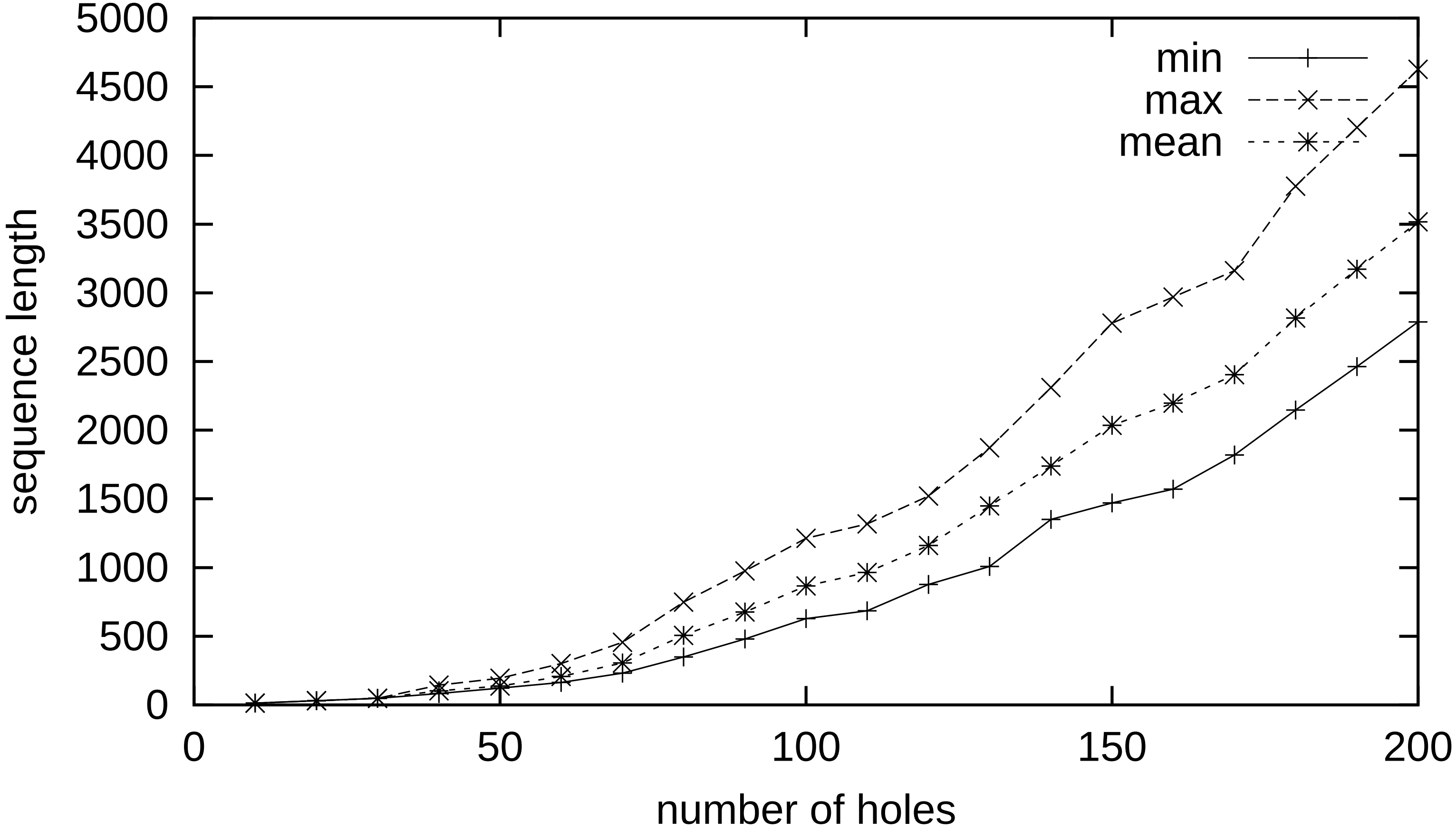
Swiss Cheese Overlap Graph ($n = 50$)



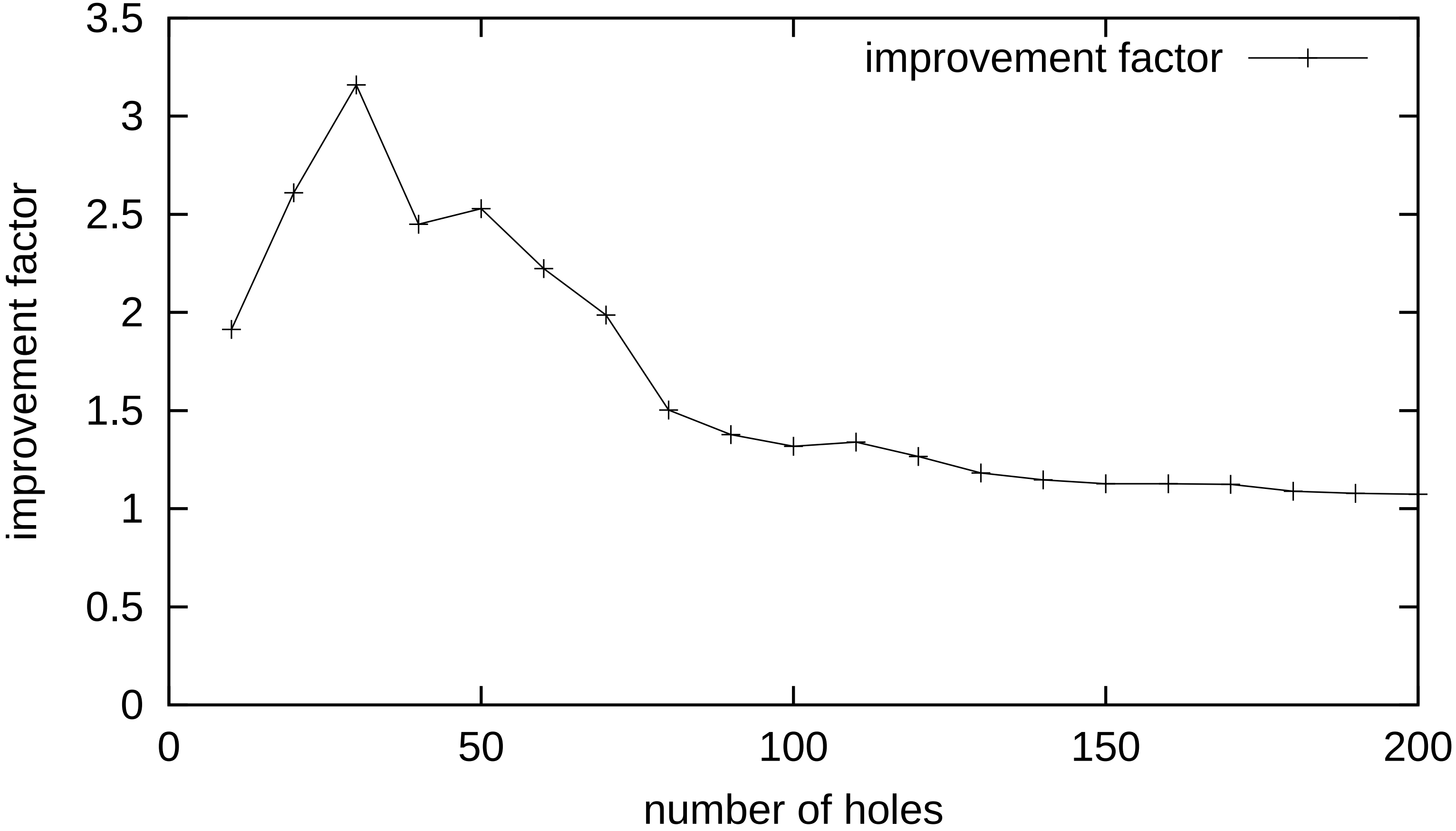
Swiss Cheese Overlap Graph ($n = 100$)

Worst case scenario for SCS CSG Rendering Algorithm.

Overlap Graph Subtraction Sequences

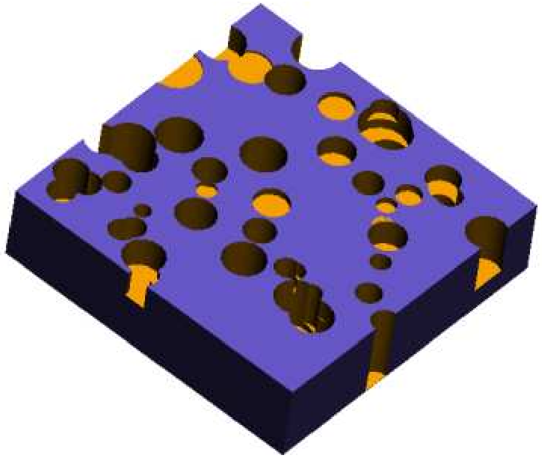


Overlap Graph Advantage

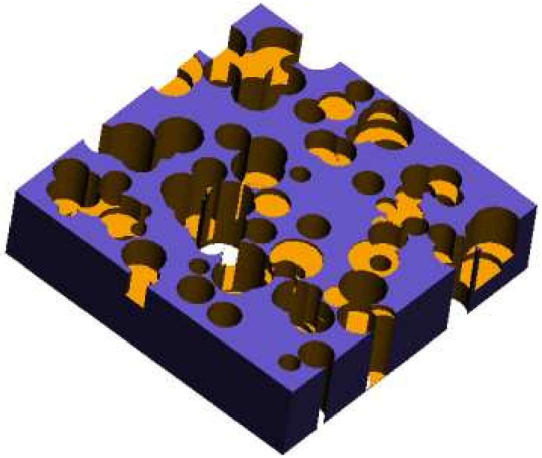


3-Axis Drilling Experiment

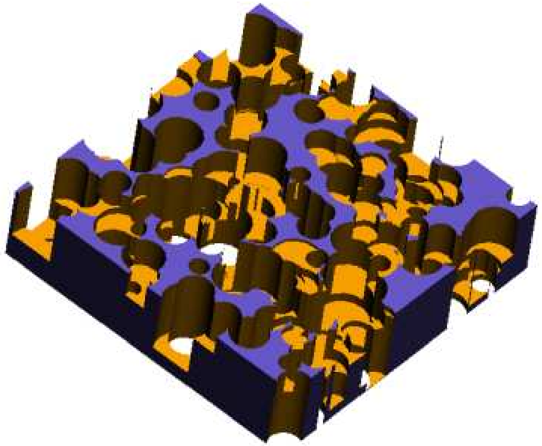
Cylindrical holes positioned and scaled randomly.



50 Subtracted Holes

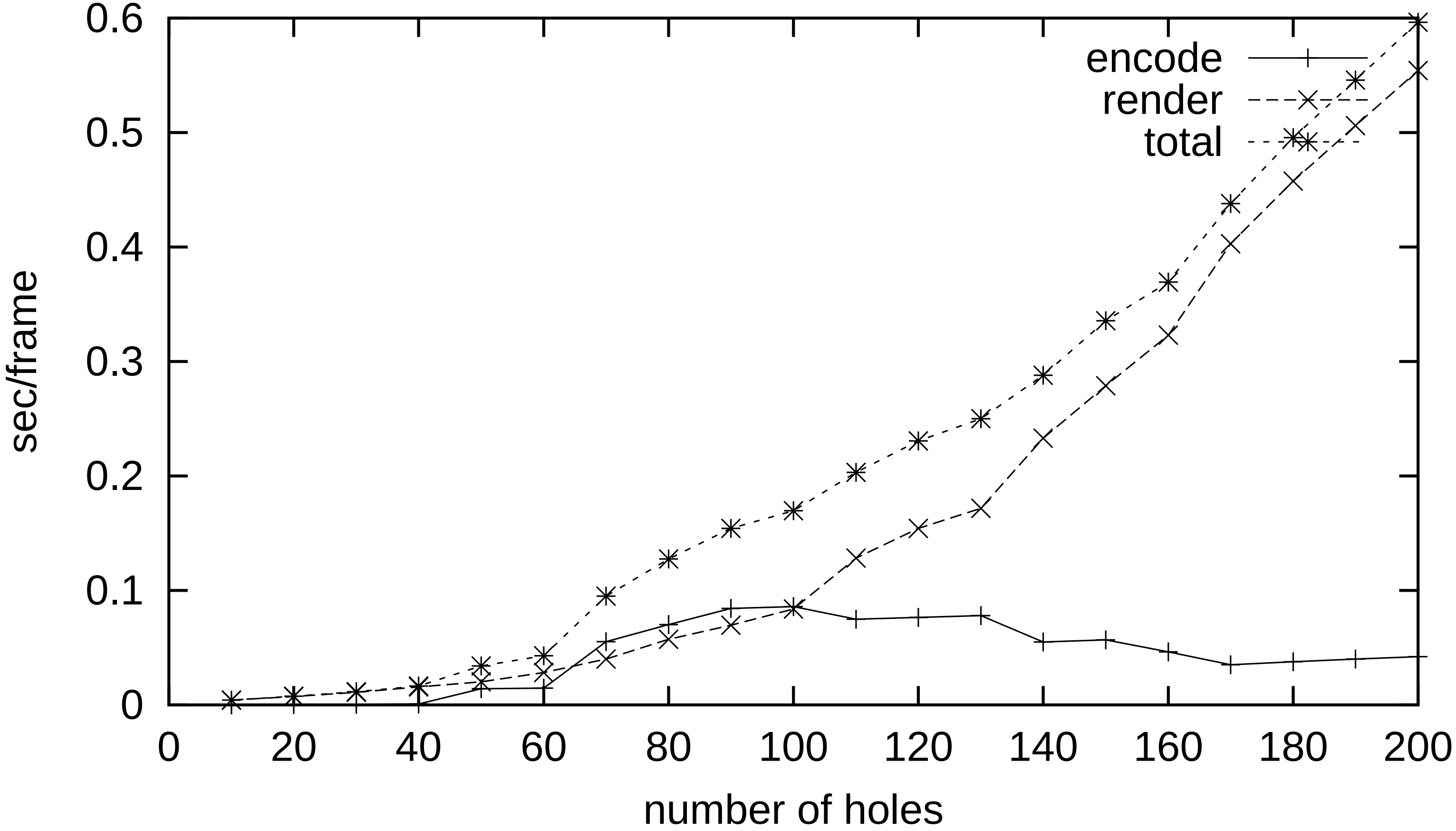


100 Subtracted Holes

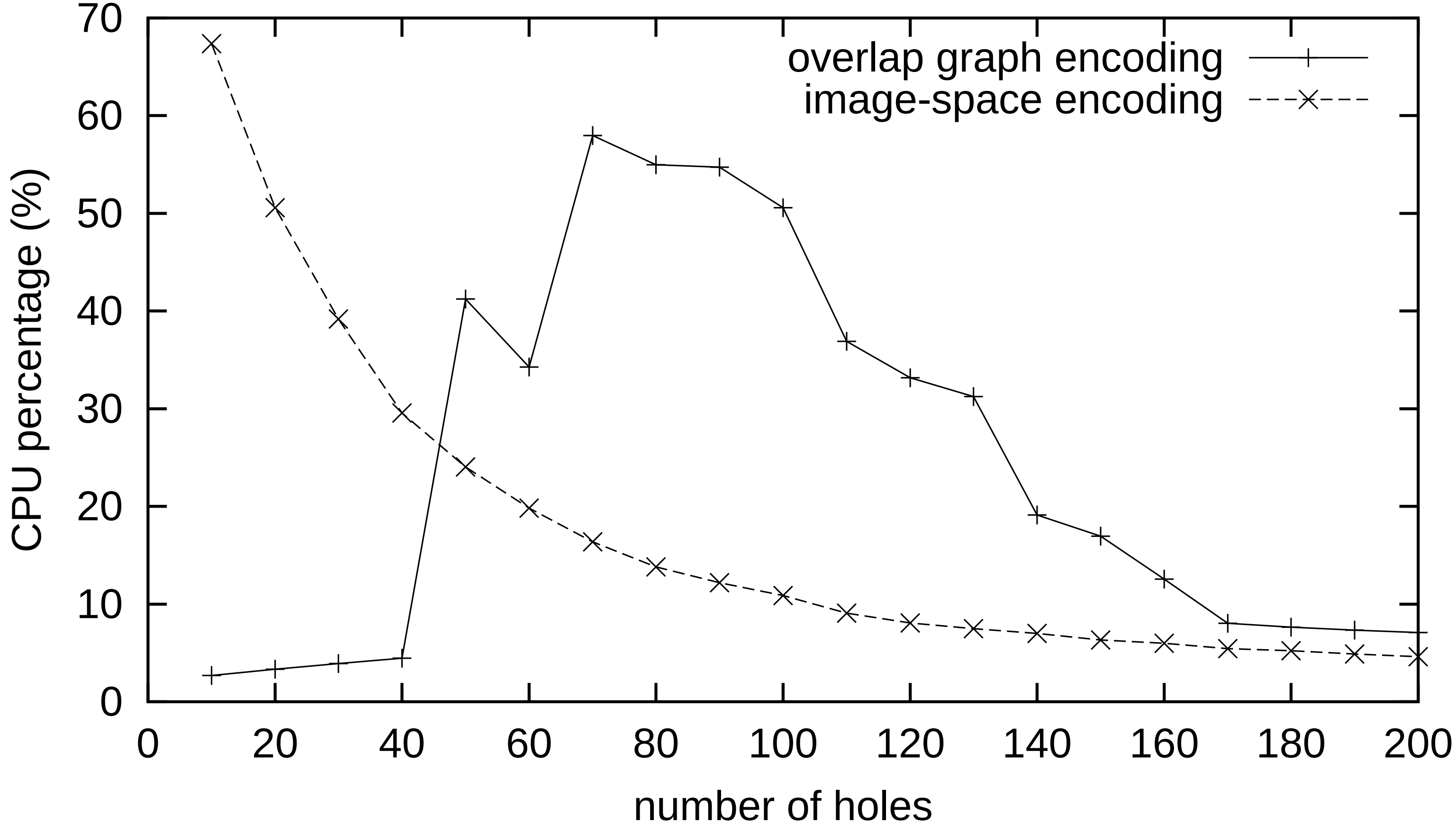


200 Subtracted Holes

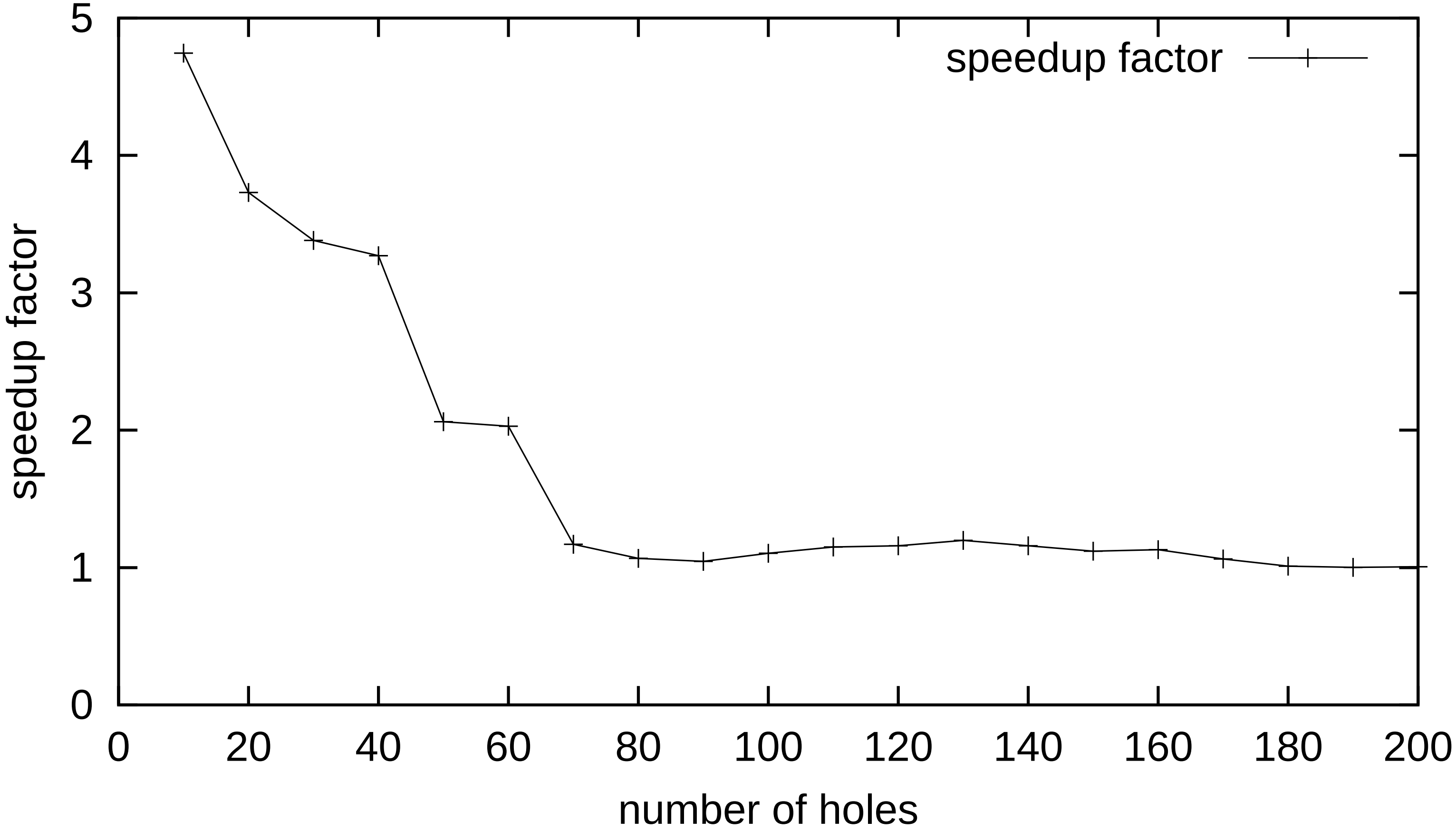
3-Axis Performance



Overlap Graph Processing



Overlap Graph Advantage



Conclusion

- Sequenced Convex Subtraction algorithm for CSG Rendering
- Overlap Graph subtraction sequences
  - Object-space spatial overlap information
  - Analysis of graph for connectivity
  - Performance improvements for sparse graphs
  - In worst case, no worse than previous methods

Further Work

- Depth complexity sampling bottleneck
- Overlap Graph techniques
  - Adjacency nodes
  - Directed Overlap Graph sequences
- Emerging graphics hardware features (extensions, shaders, etc.)

Swiss Cheese Demo

3Axis Demo