Improved CSG Rendering using Overlap Graph Subtraction Sequences

Nigel Stewart[*], Geoff Leach[†]
School of Computer Science and Information Technology
RMIT University
Melbourne, Australia

Sabu John[‡]
Department of Mechanical and Manufacturing Engineering
RMIT University
Melbourne, Australia

Abstract

The *Sequenced Convex Subtraction* (SCS) algorithm for *Constructive Solid Geometry* (CSG) sequentially subtracts convex volumes from the z-buffer. The performance of the algorithm is determined by the length of the subtraction sequence used. View-independent subtraction sequences are $O(n^2)$ in length. These can be reduced to $O(kn)$ if the maximum depth complexity k, which ranges between 1 and n, is known or can be determined.

We present an improvement to subtraction sequence generation which uses object space overlap information to give $O(n)$ length sequences in the best case and (unchanged) $O(n^2)$ sequences in the worst case. The approach is based on what we term an *overlap graph*. We also discuss a unifying approach combining overlap graph based processing with the Sequenced Convex Subtraction (SCS) CSG rendering algorithm. Finally, we present experimental results which show performance improvements, depending on the spatial arrangements of objects.

CR Categories: I.3.5 [Computer Graphics]: Geometric Algorithms, Languages, and Systems I.3.3 [Computer Graphics]: Display Algorithms I.3.7 [Computer Graphics]: Visible Line/Surface Algorithms

Keywords: CSG, rendering algorithm, graph algorithm

1 INTRODUCTION

The *Sequenced Convex Subtraction* (SCS) algorithm[Stewart et al. 2000; Stewart et al. 2002] is an image-space approach to CSG rendering. The algorithm utilises an $O(n)$ algorithm for intersection of convex objects and an $O(n^2)$ algorithm for subtraction of convex objects[Stewart et al. 2002]. Previous image-space CSG rendering algorithms[Epstein et al. 1989; Goldfeather et al. 1989; Wiegand 1996] rely on multiple z-buffers, extended z-testing or z-buffer copying. The SCS algorithm aims to maximise performance by increased utilisation of high-performance polygon rasterisation, which is typically a fast path, rather than z-buffer copying, which is typically a slow path[Wiegand 1996; Stewart et al. 1998; Stewart et al. 2000].

[*]e-mail: nigels@nigels.com
[†]e-mail: gl@cs.rmit.edu.au
[‡]e-mail: Sabu.John@rmit.edu.au

In this paper we present an approach which results in faster subtraction of large numbers of convex objects from the z-buffer. Object-space intersection detection (spatial overlap) is used as a means of producing shorter subtraction sequences. In the best case subtraction sequences of length $O(n)$ can be determined based on the spatial arrangement of objects — an improvement over the $O(n^2)$ sequences which would otherwise be needed. In the worst case the subtraction sequence length remains the same.

The *overlap graph* which stores spatial intersection information and is the basis of our improved approach is introduced in Section 2. Properties of the overlap graph we use are discussed in Section 3, including methods for encoding subtraction sequences. In Section 4 a complete algorithm is presented for encoding subtraction sequences from overlap graphs. Experimental results and a performance comparsion are presented in Section 5. Finally, concluding remarks are given in Section 6.

2 OVERLAP GRAPH

The SCS algorithm requires a subtraction sequence for each separate CSG product. Any CSG tree can be represented as a *union of products* — termed *sum-of-products*[Goldfeather et al. 1989] form. CSG tree normalisation is the process of converting a CSG tree to sum-of-products form. CSG products consist only of intersections and subtractions.

We use what we term an *overlap graph* to represent the spatial relationship of the objects in a CSG product. Nodes in the graph correspond to shapes or objects while edges in the graph indicate spatial overlap (that is, spatial intersection) between objects. Nodes and edges are removed from the graph as the graph is processed into a subtraction sequence. Aspects of this processing are described in Section 3.

Our approach does not require a perfect overlap graph. Extra edges in the overlap graph are permissible but may result in longer subtraction sequences and thereby degrade the performance of the SCS CSG rendering algorithm. Extra edges (false positives) in the overlap graph occur when overlap detection is based on bounding volumes. We use bounding volumes to build the overlap graph. More efficient approaches for intersection detection are available but overlap graph construction is not the focus here.

2.1 Graph Theory Background

In this section we introduce graph theory terminology and notation used in our work[Diestal 2000].

A *graph* $G = (V, E)$ is a set of *nodes* V and a set of *edges* E. Edges connect pairs of nodes in the graph. The *order* of a graph, denoted $|G|$ is the number of nodes in a graph. The number of edges in a graph is denoted $||G||$.

If $v \in V$, $e \in E$ and $v \in e$, then the node v is an *end* of edge e, edge e is *incident* on v, and e is an edge at node v. Two nodes x, y are *adjacent* if xy is an edge of G. Two edges are *adjacent* if they have an end in common. The *degree* of a node $d(v)$ is the number of edges incident at v. Nodes of degree zero are termed *isolated*. Nodes of degree one are called *leaves*.

A *path* is a graph $P = (V,E)$ linking two end nodes via intermediate nodes and edges. The degree of the end nodes in a path is 1 and the degree of the intermediate nodes is 2. The *length* of a path is the number of edges. A *cycle* is a graph $C = (V,E)$ connecting the nodes of the cycle V into a loop along the edges E. The degree of the nodes in a cycle is 2. The *length* of a cycle is the number of nodes or edges. A graph containing a cycle is *cyclic*. A cyclic graph consisted of only one cycle is called a *ring*. A graph containing no cycles is *acyclic*.

A graph is *connected* if every pair of nodes is connected by a path in G. Otherwise, the graph is *disconnected*. If U is a set of nodes, $G-U$ is obtained by deleting all the nodes in $U \cap V$ and their incident edges.

2.2 Implementation

Our overlap graph is implemented in C++ as an adjacency-list using the standard library vector and map templated containers. Use of a sparse representation is based on the assumption that there are generally few edges at each node. Sparse graphs are more likely to result in short subtraction sequences. We expect that real-world applications typically result in sparse graphs.

The implementation uses a std::vector of std::maps, one map for each node in the graph. Each map is a balanced tree of edges incident on a particular node. Each edge is stored at both end nodes. The class interface supports edge addition and node removal. Node degree and edge list queries are efficient.

Our own shape intersection testing routines have been used for overlap graph initialisation. In principle one of the available collision detection libraries[Cohen et al. 1995] could be used as an alternative. Optimisation of this aspect was not pursued in this investigation due to our focus on per-frame performance. Overlap graph construction is usually a once off pre-processing step.

3 OVERLAP GRAPH PROCESSING

The overlap graph contains information about the spatial relationship of the objects in a CSG tree. This section presents several properties of overlap graphs that can be used to construct subtraction sequences for the SCS CSG rendering algorithm.

The aim of overlap graph processing is to embed all necessary sequences in a combined subtraction sequence. An optimal subtraction sequence is in front-to-back order, requiring a linear sequence of subtraction steps to achieve the correct result. The focus of this work is to utilise overlap information rather than sorting. The aim is to embed every graph path in a subtraction sequence of minimal length. Each path represents a potential sequence of subtractions necessary for ensuring the correctly rendered result.

3.1 Intersected Objects

For a CSG product to be non-empty, all intersected objects in the product must overlap all other intersected objects in the product. If any pair of intersected objects in the product do not overlap then the whole product is empty and no rendering is required.

Algorithm 1 Intersected Objects Check

```
$empty \leftarrow false$
**for all** pairs of intersected objects: $i$ and $j$ **do**
    **if** $i$ and $j$ are not overlapping **then**
        $empty \leftarrow true$ {CSG product is empty}
        stop {No further processing required}
```

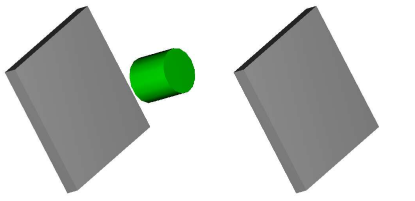


Figure 1: External Subtracted Object

3.2 External Subtracted Objects

Surfaces of subtracted objects in the CSG product must be inside all intersected objects in the product to be visible. Subtracted objects in the product not overlapping all intersected objects in the product are external and can be omitted from the subtraction sequence.

In the example in Figure 1, a cylinder is subtracted from a rectangular block. The cylinder and block do not overlap, so the cylinder need not be subtracted. The resulting subtraction sequence for the set of external nodes is empty. This process is analogous to view-frustum culling — only subtracted objects overlapping particular areas of interest need to proceed to subsequent processing steps.

Algorithm 2 External Subtracted Objects

```
**for all** subtracted objects: $i$ **do**
    **for all** intersected objects: $j$ **do**
        **if** $i$ and $j$ are not overlapping **then**
            remove $i$ from overlap graph {$i$ is external and not visible}
```

3.3 Leaf Nodes

Intersected objects in the CSG product are not included in subtraction sequences. The previous two tests make use of the spatial overlap of intersected objects in the product. Intersected objects are removed from the overlap graph at this stage since there is no further use for this information.

Leaf nodes are those having a degree of one (connected by one edge only) — and represent subtracted objects which only overlap one other subtracted object in the overlap graph. Leaf node removal can result in new leaf nodes, a cyclic graph, isolated nodes, or an empty graph. A set of removed leaf nodes is referred to as a *trim*. The overlap graph is trimmed until no further trimming is possible, resulting in a set of trims: $T_1, T_2, ..., T_n$. Repeated trimming results in either a cyclic graph, isolated nodes, or an empty graph.

In the example in Figure 2, three cylinders are subtracted from a rectangular block. The two outer cylinders are trimmed as leaves in the first pass.

Subtraction sequence encoding is based on the observation that necessary sequences of subtraction proceed from the outer trims

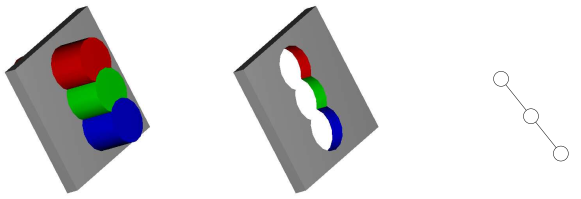


Figure 2: Leaf Nodes

Algorithm 3 Leaf Nodes

```
pass ← 1
while leaf nodes exist do
    $T_{pass}$ ← leaves
    remove $T_{pass}$ from overlap graph
    pass ← pass + 1
```

towards the inner trims and then back out towards the outer trims. Denoting $S_{cyclic}(G_{cyclic})$ as the subtraction sequence for the graph resulting from leaf trimming, the subtraction sequence S_{leaves} is: $T_1, T_2, ..., T_n \cdot S_{cyclic}(G_{cyclic}) \cdot T_n, ..., T_2, T_1$.

In the example in Figure 2, $T_1 = A, C$ and $S_{cyclic} = B$. The combined subtraction sequence is: *ACBAC*. Each leaf node appears in the subtraction sequence twice, resulting in $O(n)$ length subtraction sequences for completely acyclic graphs.

3.4 Ring Graphs

Repeated leaf node trimming results in either a cyclic graph or an empty graph. Cyclic graphs forming a ring are identified and encoded individually. In the example in Figure 3, four cylinders subtracted from a rectangular block form a ring in the overlap graph.

A ring is formed by a set of nodes with degree two connected with edges that form a loop. The following algorithm is used to find a ring in an overlap graph:

Algorithm 4 Ring Graphs

```
for all nodes of degree two: i do
    j ← overlapping node of i
    $R_0$ ← i
    $R_1$ ← j
    k ← 1
    while degree of $R_k$ is two do
        next ← unvisited overlapping node of $R_k$
        if next is $R_0$ then
            remove $R_0, .., R_k$ from overlap graph
            $G_{ring}$ ← $R_0, .., R_k$
            stop
        k ← k + 1
        $R_k$ ← next
```

Subtraction sequences for ring graphs need to include all clockwise and anti-clockwise traversals of the ring. All traversals in one direction are encoded in the sequence $R_0R_1...R_nR_0R_1...R_{n-1}$, and in the other direction $R_n...R_1R_0R_n...R_1$. The length of each of these sequences is $2n-1$. Combining the sequences in each direction, the length of ring subtraction sequences is $4n-2$.

Graph nodes processed as rings appear in the subtraction sequence up to 4 times, resulting in subtraction sequences of $O(n)$ length.

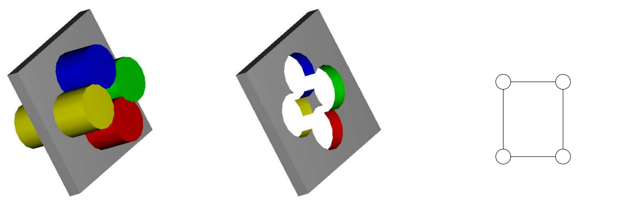


Figure 3: Ring Graph

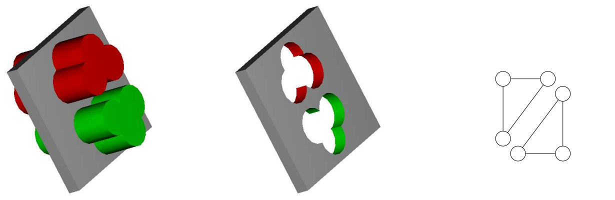


Figure 4: Disconnected Graphs

3.5 Disconnected Graphs

An overlap graph may be disconnected, that is, composed of separate connected components. Each connected component is treated individually when converted to a subtraction sequence. There is no need to embed sequences between disconnected portions of the overlap graph.

Individual connected components can be identified by starting from any node and traversing all the known edges until no new nodes can be found. This can be implemented as either a depth-first or breadth-first traversal.

Each component can be encoded separately and combined into a concatenated subtraction sequence without concern for the order. Separate connected components in the overlap graph have no interdependence. Figure 4 illustrates six subtracted objects forming a disconnected graph with two connected components.

Isolated overlap graph nodes (those with a degree of zero) can be treated as trivial connected components. The subtraction sequence for an isolated node is simply the node itself — the object need only be subtracted once to ensure the correctly rendered result.

3.6 Cyclic Graphs

Cyclic graphs that are not rings are encoded as either $O(n^2)$ view-independent or $O(kn)$ view-dependent 'image-space' subtraction sequences. These encoding algorithms have been described previously[Stewart et al. 2000; Erra et al. 2001; Stewart et al. 2002].

In the example in Figure 5, the four subtracted cylinders are overlapping all the other cylinders. None of the nodes are external to the block or are leaf nodes. Also, the cylinders do not form a ring. In this case either a $O(n^2)$ view-independent or a $O(kn)$ view-dependent subtraction sequence must be used.

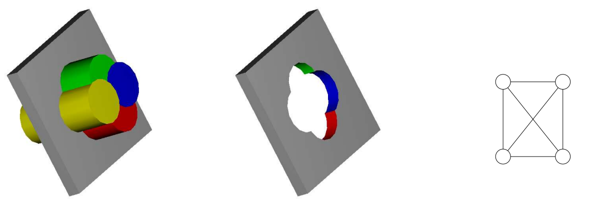


Figure 5: Cyclic Graph

4 COMPLETE ALGORITHM

4.1 Overview of SCS

The SCS CSG rendering algorithm operates in four general phases:

- CSG tree normalisation[Goldfeather et al. 1989]
- Overlap graph construction
- Subtraction sequence encoding
- Rendering[Stewart et al. 2002]

Typically, the last two steps are performed for every frame. If the viewing direction does not change between frames, then only the rendering phase needs to be repeated. This section focuses on the third phase: subtraction sequence encoding. This is where the improvements for CSG rendering from using object-space overlap information arise.

4.2 SCS Overlap Graph Encoding Algorithm

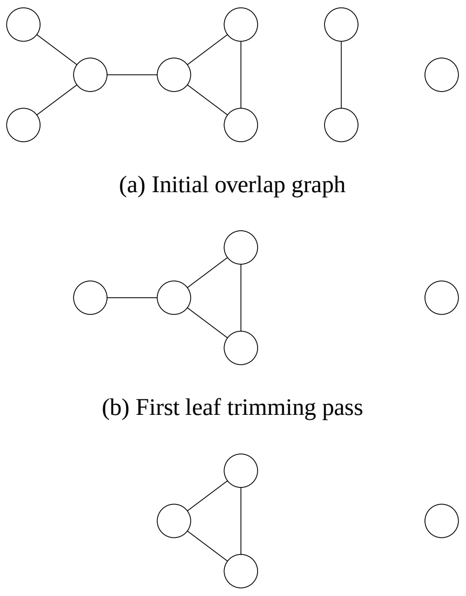
Algorithm 5 combines the methods in Section 3 to produce a subtraction sequence S given an overlap graph G and the viewing direction.

Algorithm 5 Overlap Graph Subtraction Sequence Encoding

```
{Empty intersection}
for all pairs of intersected objects: $i$ and $j$ do
    if $i$ and $j$ are not overlapping then
        $S \leftarrow \phi$
        stop
{External subtracted objects}
for all subtracted objects: $i$ do
    for all intersected objects: $j$ do
        if $i$ and $j$ are not overlapping then
            remove $i$ from overlap graph
{Leaf nodes}
for all leaf trims: $T_i$ do
    remove $T_i$ from overlap graph
{Ring graphs}
for all ring sub-graphs: $G_{ring}$ do
    remove $G_{ring}$ from overlap graph
    $S_{inner} \leftarrow S_{inner} \cdot S_{ring}(G_{ring})$
{Cyclic connected graphs}
for all connected cyclic graph: $G_{cyclic}$ do
    remove $G_{cyclic}$ from overlap graph
    $S_{inner} \leftarrow S_{inner} \cdot S_{cyclic}(G_{cyclic})$
{Combined subtraction sequence}
$S \leftarrow T_1 \cdot T_2 \cdot \ldots \cdot T_n \cdot S_{inner} \cdot T_n \cdot \ldots \cdot T_2 \cdot T_1$
```

4.3 Example

Subtraction sequence encoding is illustrated in Figure 6. Nine subtracted objects are tested for mutual intersection and form the overlap graph in Figure 6(a). Intersected nodes are not shown — empty intersection and external subtracted object tests have already been applied. The result of the first leaf trimming pass is illustrated in Figure 6(b). Four leaf nodes are removed in total, which form the


(a) Initial overlap graph

(b) First leaf trimming pass

(c) Second leaf trimming pass

Figure 6: Overlap Graph Sequence Encoding Example

prefix and postfix of the combined subtraction sequence. The second trimming pass removes an additional leaf node as illustrated in Figure 6(c). Now that no further leaves exist in the overlap graph, cyclic connected graphs are considered. The ring graph of size three is encoded as the first part of the inner subtraction sequence. Finally, the remaining (isolated) node is added to the inner subtraction sequence. In this case the length of the combined subtraction sequence is $O(n)$ and no depth complexity sampling is necessary.

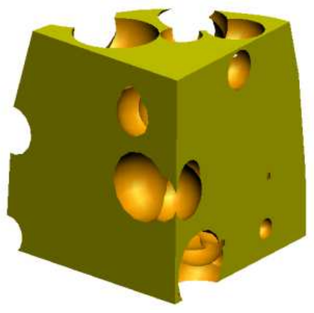
The combined sequence is formed by surrounding the encoded cyclic graphs with the leaf trims:
$T_1 \cdot T_2 \cdot S_{ring}(G_{ring}) \cdot S_{isolated}(G_{isolated}) \cdot T_2 \cdot T_1$

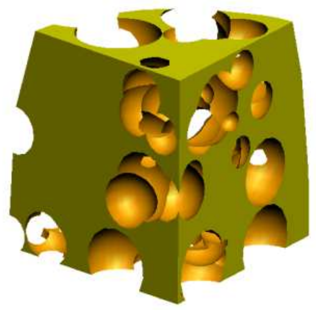
5 EXPERIMENTAL RESULTS

Results for two experiments are presented in this section. The length of subtraction sequences for image-space and overlap graph encoding algorithms are compared by using a procedural swiss cheese CSG model. In the second experiment, a 3-axis milling simulation is used to examine the real-world performance of subtraction sequence encoding and rendering.

Our previous CSG rendering implementation[Stewart et al. 2002] has been extended to include overlap graph intersection testing and subtraction sequence generation. These are implemented as a C++[Stroustrup 1997] class library using the standard C++ template library[Josuttis 1999], OpenGL[Woo et al. 1999; Shreiner 1999] and GLUT[Kilgard 1996]. The experimental platform is a 1.6GHz Intel Pentium 4, 256MB RAM, RedHat Linux 7.3 and NVIDIA GeForce4 Ti 4200 graphics hardware. The implementation is also portable to Windows and other UNIX platforms. Experiments were conducted using a 800x600 pixel OpenGL window, 24 bit z-buffer and 8-bit stencil buffer.

Overlap graph construction and performance are not the focus of this work. Our implementation using spatial subdivision was sufficient for the purpose of the experiments reported here.


(a) n=50


(b) n=100

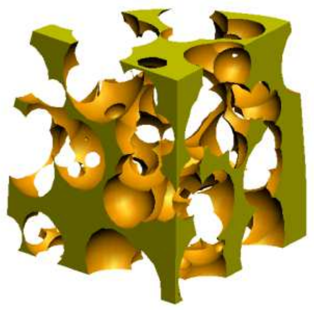
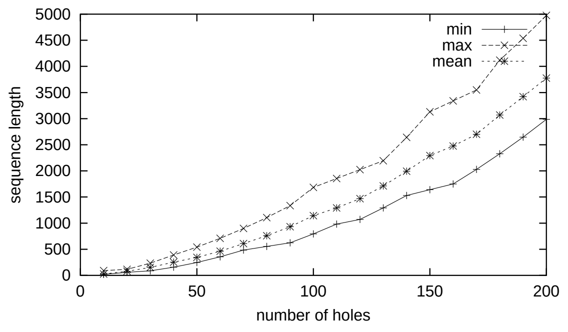
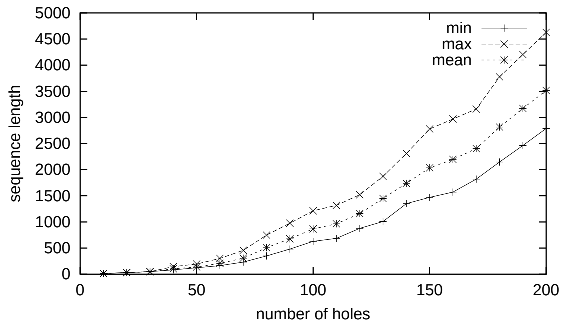

(c) n=200

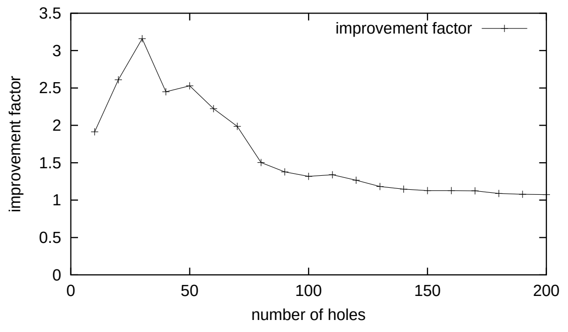
Figure 7: Procedural Swiss Cheese



(a) Image-space subtraction sequences



(b) Overlap graph subtraction sequences



(c) Overlap Graph improvement factor

Figure 8: Swiss Cheese Subtraction Sequences

5.1 Swiss Cheese Sequence Length

The swiss cheese CSG model is generated procedurally. The user supplies a specification of the number and maximum size of holes, and a seed for the random number generator. In this experiment, the maximum size of holes and random number sequence are fixed, and the number of holes varied between 10 and 200. The model is illustrated in Figure 7 with 50, 100 and 200 holes.

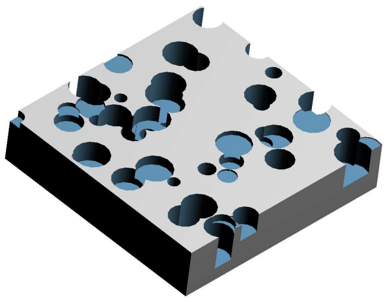
Table 1 lists the minimum, maximum and average subtraction sequence lengths for swiss cheese with up to two hundred holes. The sequence lengths were obtained by observing one thousand random viewing directions. Image-space subtraction sequence lengths are plotted in Figure 8(a), and overlap graph object-space sequence lengths are plotted in Figure 8(b). The improvement of the overlap graph object-space sequence over the image-space sequence for the mean sequence length is plotted as an improvement factor in Figure 8(c).

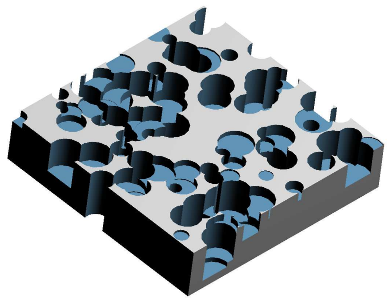
These results indicate that object-space overlap graph subtraction sequences are consistently shorter. The minimum, maximum and average case characteristics are similar. In Figure 8(c) for small *n*, sequence length improvement factors of between two and three are observed. As *n* approaches 80, the relative advantage drops dramatically and approaches an asymptote at unity as *n* increases. This reflects the fact that as the number of holes increases the overlap graph becomes more dense and cyclic and the opportunity for leaf node pruning diminishes. Overlap graph subtraction sequences will never be longer, since in the worst case the image-space encoding algorithm is used.

To summarise, the swiss cheese experiment confirms the advantage of overlap graph subtraction sequence encoding with substantially shorter sequences possible for sparse overlap graphs.

n	Image-space sequence			Overlap graph sequence		
	min	max	mean	min	max	mean
20	58	115	78.3	30	30	30.0
40	157	391	252.4	83	143	103.0
60	355	709	462.9	164	301	208.3
80	554	1107	759.6	349	749	505.6
100	793	1684	1143.0	628	1212	867.1
120	1072	2024	1468.4	876	1520	1159.7
140	1530	2642	1993.4	1351	2311	1737.9
160	1750	3340	2476.2	1570	2970	2196.9
180	2328	4118	3068.5	2146	3776	2816.7
200	2986	4976	3777.0	2788	4628	3517.2

Table 1: Swiss Cheese sequence length


(a) n=50


(b) n=100

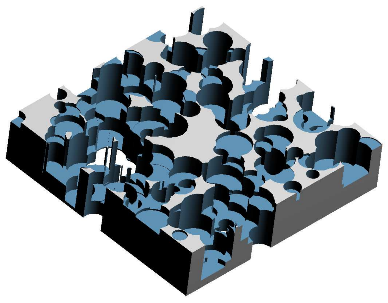

(c) n=200

Figure 9: Simulated 3-Axis Drilling

5.2 3-Axis Drilling Timing Results

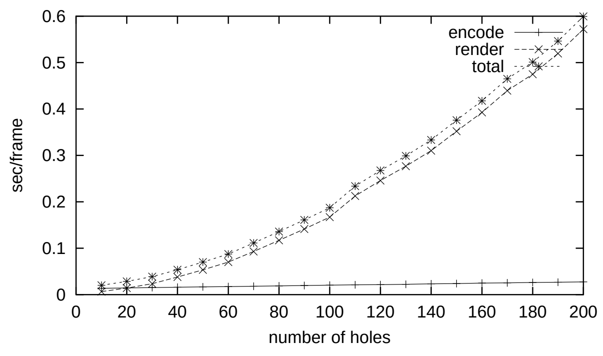
This experiment considers a simulated 3-axis drilling scenario. User specified parameters for the number and maximum size of holes are used to randomly generate drilled holes of varied radius and position. The model is illustrated in Figure 9 with 50, 100 and 200 holes.

This experiment focuses on time rather than sequence length and examines the processing overhead of object-space overlap graph sequence encoding. Table 2 lists the encoding time and total time for 3-axis drilling models with up to two hundred holes. One thousand random viewing directions were sampled for image-space based and overlap graph based sequence encoding methods. Image-space rendering times are plotted in Figure 10(a), and overlap graph rendering times are plotted in Figure 10(b). The relative performance of overlap graph CSG rendering is plotted as a speed-up factor in Figure 10(c). The proportion of CPU time spent encoding subtraction sequences is plotted as a percentage in Figure 11.

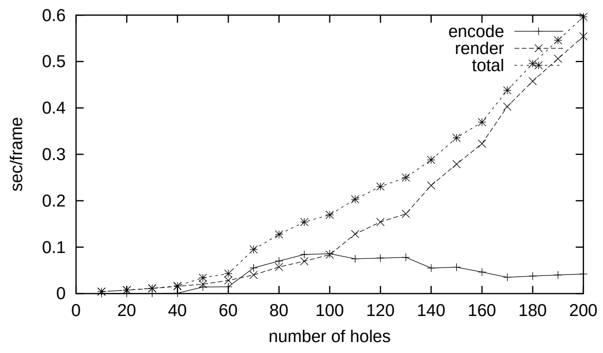
For n up to 40 all nodes are processed in leaf trimming resulting in no need for depth complexity sampling. Overlap graph sequence encoding is particularly advantageous in these cases, resulting in overall speedups of between three and four. As drill holes start forming cyclic clusters for $60 \leq n \leq 90$ execution time for overlap-graph encoding increases in both absolute and relative terms. For $n > 100$ overlap graph encoding time decreases as the model becomes increasingly dense and cyclic. As the opportunity for leaf trimming and ring finding diminish, the overall performance of both approaches converge.

	Image-space			Overlap Graph		
n	encode	render	total	encode	render	total
20	14.4	14.1	28.5	0.3	7.4	7.6
40	15.9	37.9	53.8	0.7	15.7	16.4
60	17.3	70.0	87.3	14.8	28.3	43.0
80	18.7	117.1	135.8	70.1	57.4	127.4
100	20.3	166.9	187.3	85.8	83.9	169.7
120	21.6	245.8	267.4	76.5	154.2	230.7
140	23.3	310.1	333.3	55.1	232.9	288.0
160	25.0	392.6	417.6	46.4	323.0	369.3
180	26.1	474.8	500.9	37.8	457.6	495.4
200	27.6	571.9	599.5	42.2	554.1	596.3

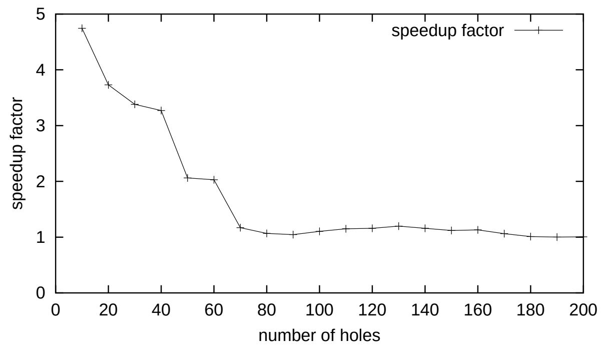
Table 2: Three-axis Timing Results (msec)



(a) Image-space encoding and total time (sec)



(b) Overlap Graph encoding and total time (sec)



(c) Overlap Graph speedup factor

Figure 10: Three-axis Timing Results

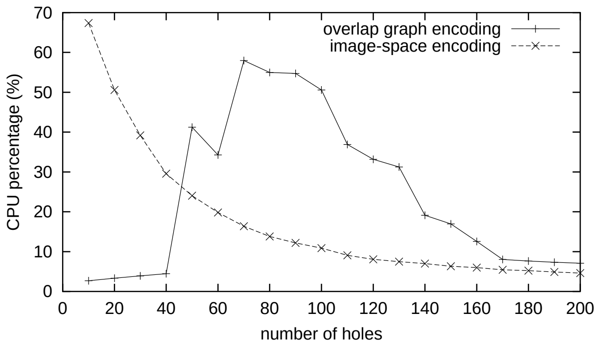


Figure 11: Three-axis Relative Encoding Time

For overlap graph sequence encoding, the fraction of time spent encoding the subtraction sequence peaks at nearly 60%, but still results in an overall speedup. In these experiments, the per-frame overlap graph encoding time is always more than offset by the performance benefit of shorter subtraction sequences. Stencil buffer copying related to depth complexity sampling has been observed to be a bottleneck at higher resolutions than 800x600. The size at which stencil buffer copying becomes the bottleneck depends on the relative rasterisation and buffer copying performance of a particular platform.

To summarise, the 3-axis drilling experiment demonstrates an overall per-frame speedup, despite some extra time spent encoding overlap graph subtraction sequences. Substantially higher frame rates are possible for sparse overlap graphs.

6 CONCLUSION

Object-space overlap graph techniques can be beneficial for decreasing the length of SCS subtraction sequences. In the best case, $O(n)$ subtraction sequences can be determined. In the worst case, $O(kn)$ subtraction sequences are still necessary. CSG trees composed of relatively sparse and evenly distributed objects especially benefit from this approach, while dense or clustered CSG trees may not.

Overlap graph sequence encoding adds an additional step to the SCS algorithm that often results in improved overall performance. Speed-up factors of up to three have been observed experimentally.

6.1 Future work

There is scope for extending these techniques to cyclic graph encoding. Graphs presenting relatively few opportunities for leaf node trimming could be approached by either grouping nodes or by considering further object-space information about adjacencies such as location and orientation.

Object-space collision algorithms such as I-COLLIDE[Cohen et al. 1995] could be combined with the SCS CSG rendering algorithm for dynamic and interactive CSG trees. Such models could be dynamic in terms of the position and shape of objects, or the arrangement of objects in the CSG tree.

Use of the stencil buffer for depth complexity sampling is a potential bottleneck for graphs consisting of many cyclic clusters. For each disconnected graph the stencil buffer is read into main memory for analysis. Alternative algorithms may need investigation.

6.2 Acknowledgements

This work was supported in part by the Co-Operative Research Center for Intelligent Manufacturing Systems & Technologies. This research arose from the C-5 collaborative research project involving ANCA Pty. Ltd., RMIT University and the CRC for IMST.

References

COHEN, J., LIN, M., MANOCHA, D., AND PONAMGI, K. 1995. I-COLLIDE: An interactive and exact collision detection system for large-scaled environments. *Proceedings of ACM Int. 3D Graphics Conference*, 189–196.

DIESTAL, R. 2000. *Graph Theory*. Sprinter-Verlag.

EPSTEIN, D., JANSEN, F., AND ROSSIGNAC, J. 1989. Z-buffer rendering from CSG: The trickle algorithm. *IBM Research Report RC 15182* (Nov).

ERRA, R., LYGEROS, N., AND STEWART, N. 2001. On minimal strings containing the elements of S_n by decimation. *Discrete Mathematics & Theoretical Computer Science AA*, 165–176.

GOLDFEATHER, J., HULTQUIST, J., AND FUCHS, H. 1986. Fast constructive solid geometry in the pixel-powers graphics system. In *Computer Graphics (Proceedings of ACM SIGGRAPH 86)*, vol. 20(4), ACM, 107–116.

GOLDFEATHER, J., MOLNAR, S., TURK, G., AND FUCHS, H. 1989. Near real-time CSG rendering using tree normalization and geometric pruning. *IEEE CG&A 9*, 3 (May), 20–28.

JOSUTTIS, N. M. 1999. *The C++ Standard Library*. Addison Wesley.

KILGARD, M. J. 1996. *The OpenGL Utility Toolkit (GLUT) Programming Interface*.

LOMBARDO, J.-C., CANI, M.-P., AND NEYRET, F. 1999. Real-time collision detection for virtual surgery. *Proceedings of Computer Animation '99*, 82–90.

SHREINER, D. 1999. *OpenGL Reference Manual*, 3rd ed. Addison Wesley.

STEWART, N., LEACH, G., AND JOHN, S. 1998. An improved Z-buffer CSG rendering algorithm. *1998 Eurographics/Siggraph Workshop on Graphics Hardware* (Aug), 25–30.

STEWART, N., LEACH, G., AND JOHN, S. 2000. A Z-buffer 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 II* (Feb), 369–372.

STEWART, N., LEACH, G., AND JOHN, S. 2002. 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 II* (Feb), 437–444.

STROUSTRUP, B. 1997. *The C++ Programming Language*, 3rd ed. Addison Wesley.

WIEGAND, T. F. 1996. Interactive rendering of CSG models. *Computer Graphics Forum 15*, 4 (Oct), 249–261.

WOO, M., NADER, J., DAVIS, T., AND SHREINER, D. 1999. *OpenGL Programming Guide*, 3rd ed. Addison Wesley.