Realtime Constructive Solid Geometry

Mikola Lysenko*
Michigan Technological University

1 Introduction

Constructive solid geometry (CSG) is a modeling technique where objects are combined using boolean operations to build up complex shapes. Fig. 1 shows the results of union, intersection and subtraction between the Stanford bunny and a toroidal knot. While many CAD systems support CSG, it remains conspicuously absent from interactive environments. Part of the problem is that current techniques are slow, unstable and complicated. This algorithm addresses these issues by improving on Naylor et al.’s tree merging algorithm [Naylor et al. 1990].

2 Tree Merging

Tree merging operates on binary space partitions (BSPs), which gives it several advantages. Unlike image based techniques, BSPs scale with scene complexity. Tree merging is numerically stable compared to boundary representations. It gives exact results, not discrete approximations like voxels, while simultaneously handling collision detection. This method builds on previous work by extending tree merging to arbitrary subdivision trees. Let $B$ be a BSP as follows:

$$\begin{align*}
B & \in \{ \text{EMPTY}, \text{FULL}, (B^+, B^-, B_H) \} \\
\text{EMPTY} & = \emptyset \\
\text{FULL} & = \mathcal{U} = \neg \emptyset \\
(B^+, B^-, B_H) & = (B^+ \cap B_H) \cup (B^- \cap \neg B_H)
\end{align*}$$

(1)

Where EMPTY and FULL represent leaf nodes, and internal nodes are tuples such as $(B^+, B^-, B_H)$. In a tuple, $B^+, B^-$ are child BSPs and $B_H$ is some partitioning set. Conversion to set expression is defined recursively as follows:

$$\begin{align*}
\text{EM} & = \emptyset \\
\text{FU} & = \mathcal{U} = \neg \emptyset \\
(B^+, B^-, B_H) & = (B^+ \cap B_H) \cup (B^- \cap \neg B_H)
\end{align*}$$

(2) (3) (4)

Given two BSPs, $A = (A^+, A^-, A_H), B = (B^+, B^-, B_H)$, a set $R$ and an operator $\oplus \in \{\cup, \cap\}$, define the merge of $A$ and $B$ as:

$$\text{merge}(A, B, R, \oplus) = R \cap (A \oplus B)$$

(5)

As Naylor et al. have shown, any CSG expression can be written in terms of $\text{merge}$; however, their algorithm required expensive BSP partitioning using polyhedra clipping. Algorithm 1 replaces partitioning with the test $A_H \cap R = \emptyset$ in lines 10 and 12. For planar BSPs $R = A_H \cap B_H \cap ...$ is always an intersection of hyperplanes, so both tests reduce to linear programming feasibility. In low dimensions Seidel’s algorithm [Seidel 1990] solves $R = \emptyset$ in $O(n)$ time, compared to the previous $O(n^3)$ polyhedra tests. The result is an asymptotic speed up by several orders of magnitude.

**Algorithm 1** Partition free tree merging

1. function merge$(A, B, R, \oplus)$
2. if $A \in \{\text{EMPTY}, \text{FULL}\}$ then
3. if $B \in \{\text{EMPTY}, \text{FULL}\}$ then
4. return $A \oplus B$
5. end if
6. swap$(A, B)$
7. end if
8. $T^+ \leftarrow \text{merge}(A^+, B, A_H \cap R, \oplus)$
9. $T^- \leftarrow \text{merge}(A^-, B, \neg A_H \cap R, \oplus)$
10. if $(T^+ = T^-) \lor (A_H \cap R = \emptyset)$ then
11. return $T^-$
12. else if $(\neg A_H) \cap R = \emptyset$ then
13. return $T^+$
14. end if
15. return $(T^+, T^-, A_H)$
16. end function

3 Conclusion

Compared to standard tree merging, Algorithm 1 is on average over 30 times faster on identical BSPs of about 6,000 polygons. Overall time cost is still dictated by the quality of BSP construction. In theory, better linear programming algorithms should further enhance performance. Finally, the technique also solves non-convex linear programming at no additional cost.

References


*e-mail: mlysenk@mtu.edu