OpenCSG: A Library for Image-Based CSG Rendering

Florian Kirsch
Introduction

- Modeling using CSG (Constructive Solid Geometry)
Introduction

- CSG requires: Primitives
  - Leaf nodes of CSG tree
  - Must be closed

- and: Boolean operations
  - Inner nodes of CSG tree
  - Union, Intersection, Subtraction
Introduction

- CSG is available in offline rendering systems
  - RenderMan
  - POV-Ray
Introduction

- CSG cannot be directly rendered with graphics hardware
- How to use CSG in real-time?

1. **Object-based CSG**
   - Calculate boundary of CSG shape mathematically, render mesh
   - Too expensive for dynamically manipulated models

2. **Image-based CSG rendering**
   - Compose final image in frame buffer
   - Suitable for interactive manipulation
   - Advanced use of graphics hardware
Algorithms

- Two basic algorithms for image-based CSG
- Goldfeather algorithm
- SCS algorithm
- Both require normalization of CSG tree to union of CSG products
- CSG product: \( \ldots ((x_1 \ominus x_2) \ominus x_3) \ldots \ominus x_n \)  
  \( (\ominus \) is intersection or subtraction)
Algorithms

- Normalization of CSG tree to union of CSG products
Algorithms

- Goldfeather algorithm: all kinds of primitives
- Handle each primitive separately
Algorithms

- Goldfeather algorithm
- Different handling of subtracted primitives
- SCS algorithm: only convex primitives
Algorithms

- Problems: Algorithms are complicated
- Seldom used in real applications
- Currently no abstraction layer available

→ OpenCSG
API: Overview

Design Goals

- Minimal interface
- Only for CSG rendering
- Possibility to use user-defined primitives
- No external dependencies except OpenGL
- ...
API: Overview

namespace OpenCSG {
    enum Operation { Intersection, Subtraction }
    class Primitive {
        public:
            Primitive(Operation, unsigned int convexity);
            virtual ~Primitive();
            void setOperation(Operation);
            Operation getOperation() const;
            void setConvexity(unsigned int);
            unsigned int getConvexity() const;
            void setBoundingBox(float minx, float miny, float minz,
                                  float maxx, float maxy, float maxz);
            void getBoundingBox(float& minx, float& miny, float& minz,
                                 float& maxx, float& maxy, float& maxz) const;
            virtual void render() = 0;
        
    }
    enum Algorithm {
        Automatic, Goldfeather, SCS
    }
    enum DepthComplexityAlgorithm {
        NoDepthComplexitySampling, OcclusionQuery, DepthComplexitySampling
    }
    void render(const std::vector<Primitive*>& primitives,
        Algorithm = Automatic,
        DepthComplexityAlgorithm = NoDepthComplexitySampling);
}
API: Overview

Basic facts

- C++ interface
- Namespace OpenCSG
- Abstract primitive class
- Render function for CSG rendering
API: Abstract primitive class

- Operation: Whether primitive in CSG product is intersected or subtracted

```cpp
enum Operation { Intersection, Subtraction };  
class Primitive {
public:
    Primitive(Operation, unsigned int convexity);
    virtual ~Primitive();
    void setOperation(Operation);
    Operation getOperation() const;
    void setConvexity(unsigned int);
    unsigned int getConvexity() const;
    void setBoundingBox(float minx, float miny, float minz,  
                       float maxx, float maxy, float maxz);
    void getBoundingBox(float& minx, float& miny, float& minz,  
                        float& maxx, float& maxy, float& maxz) const;
    virtual void render() = 0;
};
```
API: Abstract primitive class

- Convexity

```cpp
enum Operation { Intersection, Subtraction }

class Primitive {
public:
    Primitive(Operation, unsigned int convexity);
    virtual ~Primitive();
    void setOperation(Operation);
    Operation getOperation() const;
    void setConvexity(unsigned int);
    unsigned int getConvexity() const;
    void setBoundingBox(float minx, float miny, float minz,
        float maxx, float maxy, float maxz);
    void getBoundingBox(float& minx, float& miny, float& minz,
        float& maxx, float& maxy, float& maxz) const;
    virtual void render() = 0;
};
```
API: Abstract primitive class

- Bounding Box: Maximum extent of primitive in screen space
- Used for rendering optimizations

```cpp
enum Operation { Intersection, Subtraction };

class Primitive {
public:
    Primitive(Operation, unsigned int convexity);
    virtual ~Primitive();
    void setOperation(Operation);
    Operation getOperation() const;
    void setConvexity(unsigned int);
    unsigned int getConvexity() const;
    void setBoundingBox(float  minx, float  miny, float  minz,
                        float  maxx, float  maxy, float  maxz);
    void getBoundingBox(float& minx, float& miny, float& minz,
                        float& maxx, float& maxy, float& maxz) const;
    virtual void render() = 0;
};
```
API: Abstract primitive class

- render method: Renders a primitive
- Implemented by subclasses
- Don‘t set colors!

```cpp
enum Operation { Intersection, Subtraction }

class Primitive {
public:
    Primitive(Operation, unsigned int convexity);
    virtual ~Primitive();
    void setOperation(Operation);
    Operation getOperation() const;
    void setConvexity(unsigned int);
    unsigned int getConvexity() const;
    void setBoundingBox(float  minx, float  miny, float  minz,
                        float  maxx, float  maxy, float  maxz);
    void getBoundingBox(float& minx, float& miny, float& minz,
                        float& maxx, float& maxy, float& maxz) const;
    virtual void render() = 0;
};
```
API: Render function for CSG rendering

- Render function for CSG rendering
- Initializes Z-Buffer with values of CSG product

```cpp
enum Algorithm {
    Automatic, Goldfeather, SCS
};
enum DepthComplexityAlgorithm {
    NoDepthComplexitySampling, OcclusionQuery, DepthComplexitySampling
};
void render(const std::vector<Primitive*>& primitives,
             Algorithm = Automatic,
             DepthComplexityAlgorithm = NoDepthComplexitySampling);
```
API: Render function for CSG rendering

- Initializes Z-Buffer with values of CSG product
- Arguments:
  - List of primitives

```cpp
enum Algorithm {
    Automatic, Goldfeather, SCS
};
enum DepthComplexityAlgorithm {
    NoDepthComplexitySampling, OcclusionQuery, DepthComplexitySampling
};
void render(const std::vector<Primitive*>& primitives,
            Algorithm = Automatic,
            DepthComplexityAlgorithm = NoDepthComplexitySampling);
```
API: Render function for CSG rendering

- Initializes Z-Buffer with values of CSG product
- Arguments:
  - List of primitives
  - Specifier to choose CSG algorithm

```cpp
enum Algorithm {
    Automatic, Goldfeather, SCS
};
enum DepthComplexityAlgorithm {
    NoDepthComplexitySampling, OcclusionQuery, DepthComplexitySampling
};
void render(const std::vector<Primitive*>& primitives,
            Algorithm = Automatic,
            DepthComplexityAlgorithm = NoDepthComplexitySampling);
```
API: Render function for CSG rendering

- Initializes Z-Buffer with values of CSG product

- Arguments:
  - List of primitives
  - Specifier to choose CSG algorithm
  - Optimization strategy

```cpp
enum Algorithm {
    Automatic, Goldfeather, SCS
};
enum DepthComplexityAlgorithm {
    NoDepthComplexitySampling, OcclusionQuery, DepthComplexitySampling
};
void render(const std::vector<Primitive*>& primitives,
            Algorithm = Automatic,
            DepthComplexityAlgorithm = NoDepthComplexitySampling);
```
Example

- Basic shapes generated by helper library GLUT
- Stored as OpenGL display lists
- Box and Sphere

GLuint id1 = glGenLists(1);
glNewList(id1, GL_COMPILE);
glutSolidCube(1.8);
gleEndList();
GLuint id2 = glGenLists(1);
glNewList(id2, GL_COMPILE);
glutSolidSphere(1.2, 20, 20);
gleEndList();
Example

- Class derived from Primitive class
- Invokes display list in render method
- Create objects for Box and Sphere
- Append to CSG product

```cpp
class DisplayListPrimitive : public OpenCSG::Primitive {
    ... 
};
namespace OpenCSG;
DisplayListPrimitive* box=new DisplayListPrimitive(id1, Intersection, 1);
DisplayListPrimitive* sphere=new DisplayListPrimitive(id2, Subtraction, 1);
std::vector<Primitive*> primitives;
primitives.push_back(box);
primitives.push_back(sphere);
```
Example

- Call render function
- This initializes Z-Buffer with values of CSG product
- Must shade the primitives now
- Render with z-equal depth test

```cpp
OpenCSG::render(primitives,
                 Goldfeather,
                 NoDepthComplexitySampling);

glDepthFunc(GL_EQUAL);
// setup lighting and shading
box->render();
sphere->render();
glDepthFunc(GL_LESS);
```
Implementation

- Implementation as described in

*Rendering Techniques for Hardware-Accelerated Image-Based CSG*  
(WSCG 2004)
Applications

- Modeling applications
- Screenshot: RenderMan modeler Ayam
Applications

- Beyond traditional uses of CSG
  Gaming
Applications

- Beyond traditional uses of CSG

Interactive road design
Questions?

www.opencsg.org
OpenCSG: Internals

Size of invisible frame buffer

- Constant, even if used from different windows
- Important for multi-window applications
- And multipass algorithms (e.g., shadow mapping)
OpenCSG: Internals

Further optimizations

- Based on bounding-boxes
- Batching of primitives
- Scissoring