Realtime Computer Graphics on GPUs

What did not fit elsewhere

Jan Kolomazník

Department of Software and Computer Science Education
Faculty of Mathematics and Physics
Charles University in Prague

Computer Graphics Charles University

Transform Feedback

What is Transform Feedback?

- Captures output from Vertex/Tessellation/Geometry Shaders
- Bypasses the rasterizer
- Writes data directly into GPU buffer objects

When to Use It

- Particle systems
- GPU-based simulations
- Geometry manipulation and reuse
- General-purpose GPU computation

How It Works

1. Create and bind buffer(s)
2. Specify varyings before linking shader
3. Enable transform feedback mode
4. Render normally while capturing outputs
5. Disable transform feedback and use the data

Shader Example: Declaring Varyings

```
#version 330 core
out vec3 newPosition;
void main() {
   // compute newPosition
   }
```

OpenGL Code Snippet

```
const char* varyings[] = { "newPosition" };
glTransformFeedbackVaryings(program, 1, varyings, ↩
    GL_INTERLEAVED_ATTRIBS);
glLinkProgram(program);
glBeginTransformFeedback(GL_POINTS);
// draw call here
glEndTransformFeedback();
```

Modes of Operation

- `GL_INTERLEAVED_ATTRIBS` – all data in one buffer
- `GL_SEPARATE_ATTRIBS` – each varying in separate buffers

Depth Peeling

What is Depth Peeling?

- An Order-Independent Transparency (OIT) technique
- Renders transparent objects correctly without sorting
- Captures multiple depth layers via multiple rendering passes

Why Use Depth Peeling?

- Traditional alpha blending is order-dependent
- Geometry sorting fails for intersecting or dynamic shapes
- Depth peeling ensures visual correctness in transparency

Basic Idea

1. Render and capture front-most fragments
2. In subsequent passes, skip already captured fragments
3. Blend each new layer with previous ones

How It Works: Step-by-Step

- **Pass 1:** Capture nearest depth layer
- **Pass 2+:** Use previous depth to discard closer fragments
- **Final:** Blend all layers for final image

Required OpenGL Resources

- Multiple framebuffers (for color and depth)
- Shaders to compare and discard previous depth
- Blending enabled (typically premultiplied alpha)

Advantages and Disadvantages

Pros

- Accurate transparency
- No geometry sorting
- Works with intersecting objects

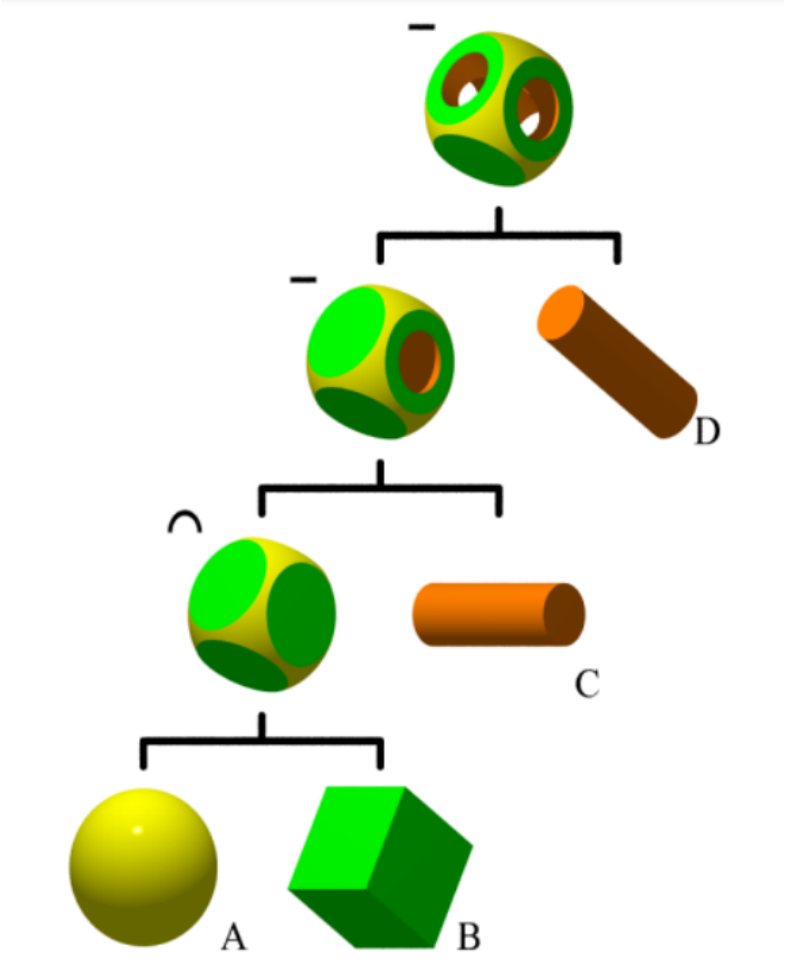
Cons

- Multi-pass rendering
- GPU memory usage
- Slower than simpler OIT methods

Constructive Solid Geometry

What is CSG?

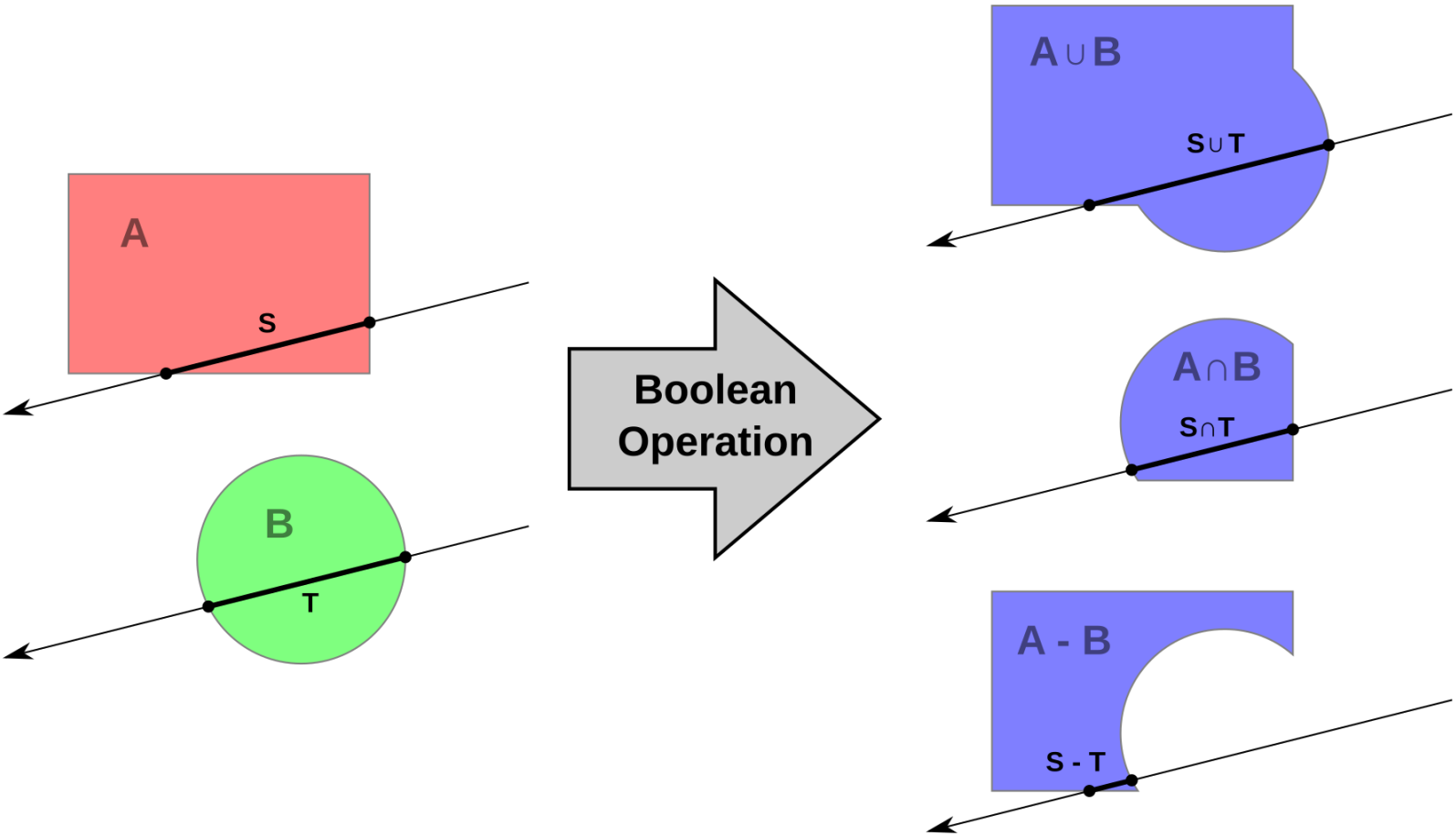
- Used by CAD applications
- Set operations on geometry primitives
  - Union
  - Intersection
  - Subtraction



How to render?

- Conversion to tri-mesh
- Raytracing
- Rasterization pipeline

RAYTRACING



Rasterization Pipeline

- Elementary solids converted to polyhedra
- set operations evaluated on the GPU
  - union is trivial (default depth-buffer based rendering)
  - intersection and subtraction: use of stencil buffer, considering front vs. back faces union is trivial (default depth-buffer based rendering)

GOLDFEATHER

- 1989: Goldfeather et al.
- normalization of a CSG tree – decomposition to union of ”products” (intersections and differences)
- implementation uses several depth-buffers and a stencil buffer (needs to copy depth-buffers)

INTERSECTION OF N SOLIDS

- init: depth = near; stencil = 0;
- passing through front faces of individual solids

```
if ( front > depth ) depth = front;
```

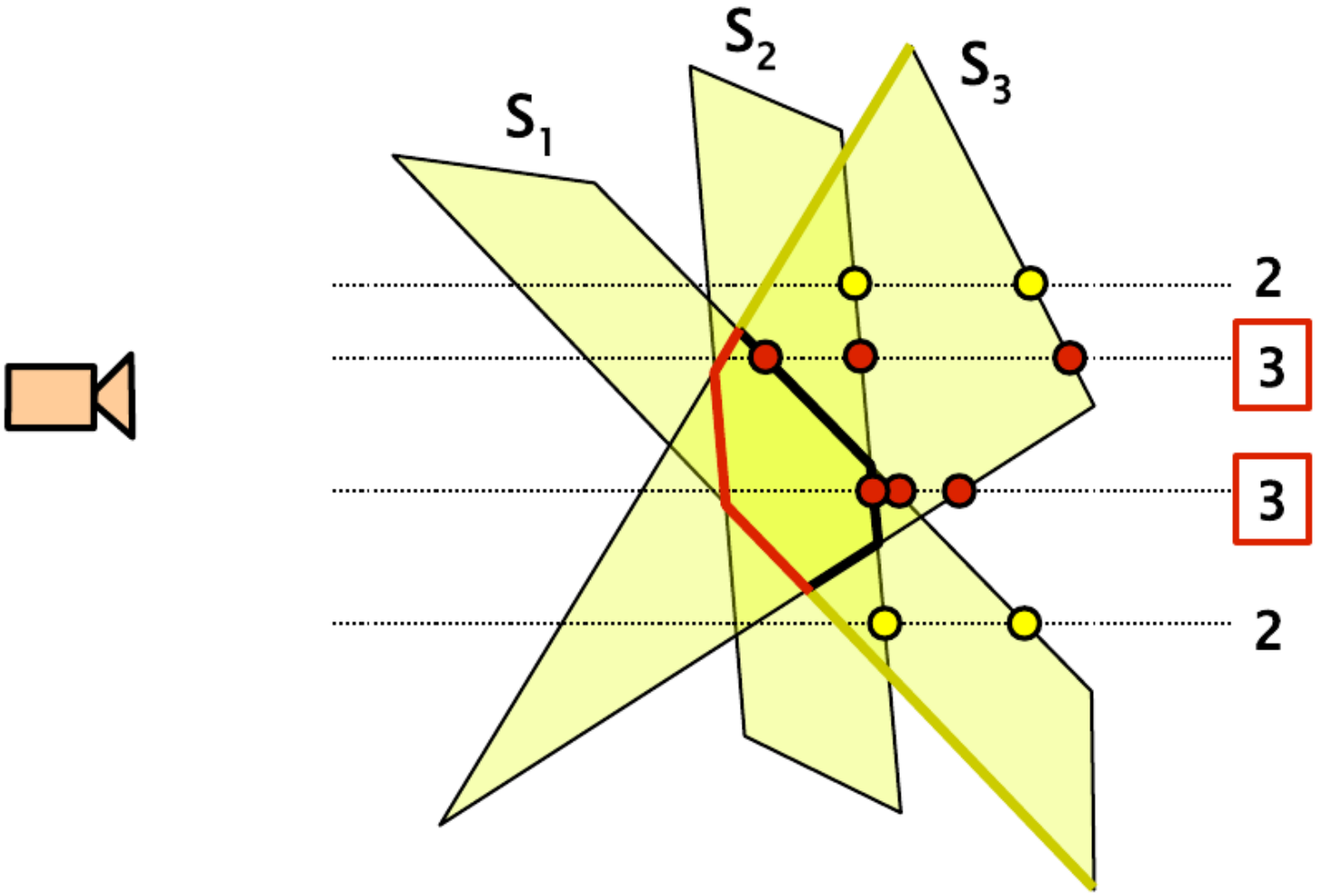
- passing through back faces (occlusions)

```
if ( back > depth ) stencil++;
```

- removing pixels with occlusion number $<$ n

```
if ( stencil < n )
{
 stencil = 0;
 depth = far;
}
```

INTERSECTION – EXAMPLE



SEQUENCED CONVEX SUBTRACTION

- 2000:Stewart et al. – Sequenced Convex Subtraction (SCS)
- does not need depth-buffer copying, complex depth-tests
- all elementary solids have to be convex

SCS Phases

1. preprocessing (CSG normalization, sorting of subtraction sequences /front-to-back/)
2. depth-buffer processing (for every product + merge)
3. final rendering to frame-buffer

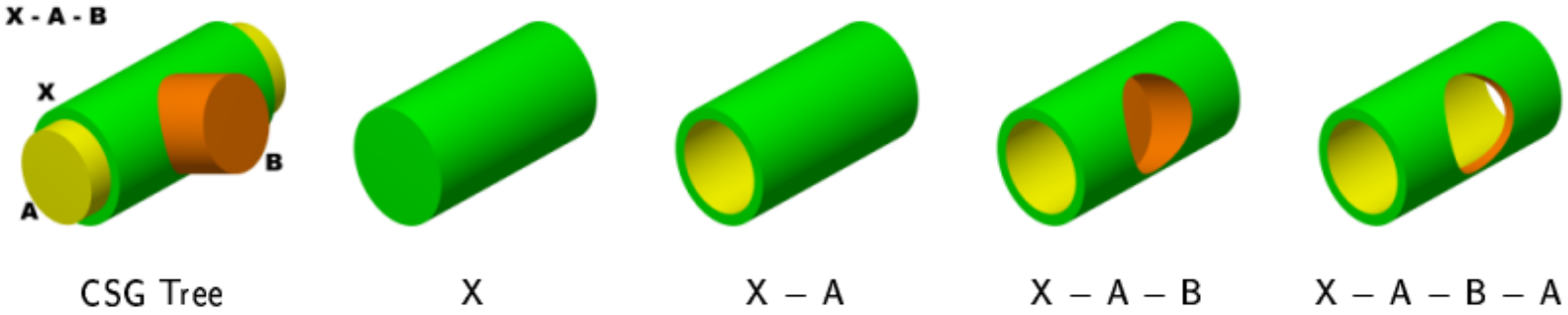
SUBTRACTING SEQUENCES

- determining correct subtracting sequence;
  - front-to-end subtraction
  - universal subtractin sequence – contains all occlusion permutations
- subtracting from the front: passing all subtr. solids

```
if ( front < depth ) stencil = 1;
else stencil = 0;
```

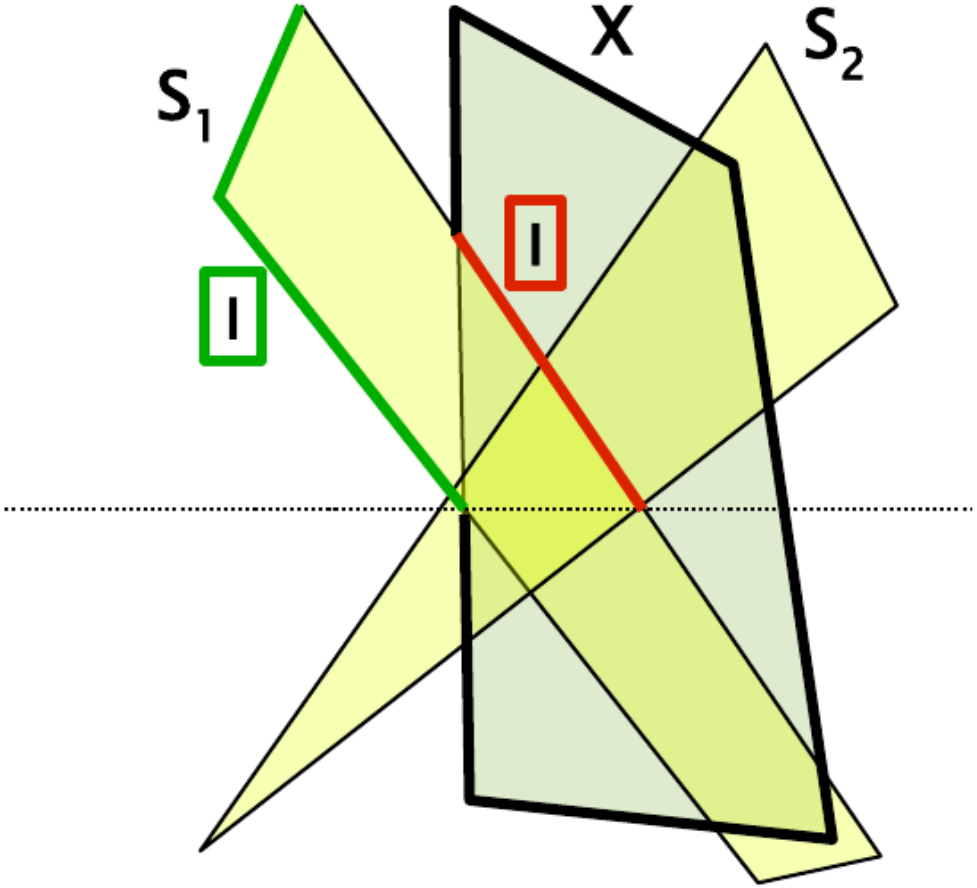
- every back-face is processed immediately as well

```
if ( back > depth && stencil == 1 )
depth = back;
```



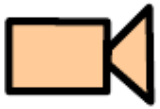
Subtraction – step I

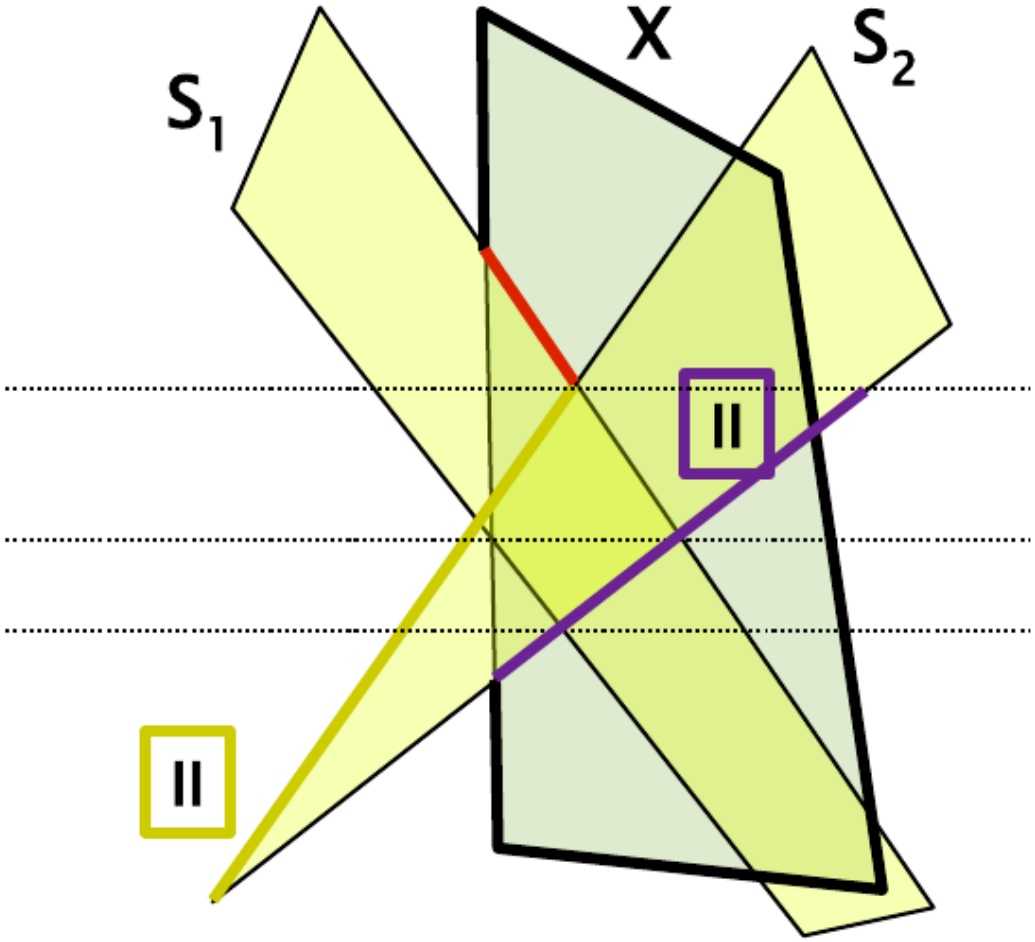
$X - S_1 - S_2 - S_1$



Subtraction – step II

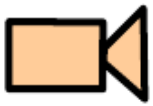
$X - S_1 - S_2 - S_1$

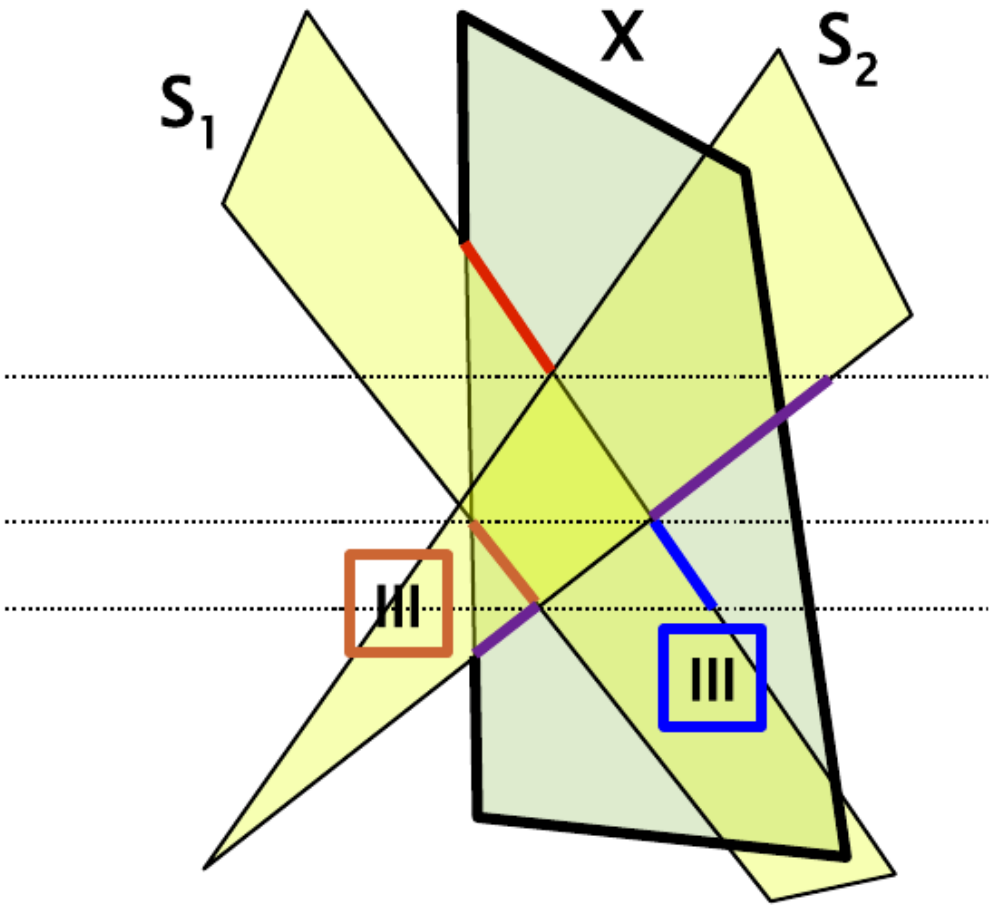




SUBTRACTION – STEP III

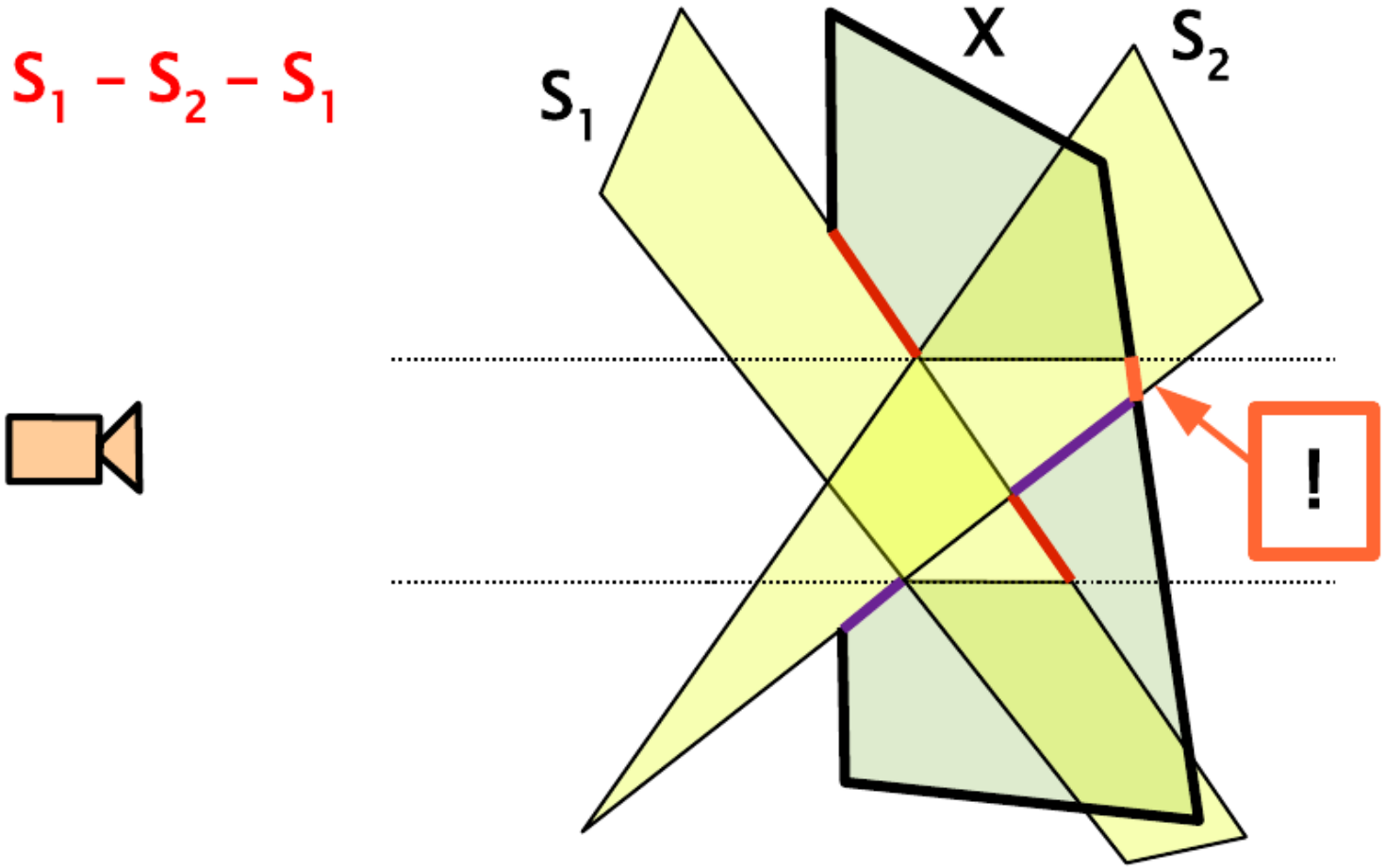
$X - S_1 - S_2 - S_1$





SUBTRACTION – RESULT

$X - S_1 - S_2 - S_1$



Completely subtracted parts

- removing parts of common intersection, which were eliminated completely
- passing through all intersection solids (back faces only – looking for empty results)
- elimination of completely subtracted parts

Merging products and final rendering

- product result = its depth buffer
- merging results of one product (i.e. union operation)
- intersections render front faces
- subtractions render back faces