SCIENTIFIC COMPUTING AND IMAGING
INSTITUTE
www.sci.utah.edu

Visualization Techniques
Volume Rendering
Isosurface Generation

Cláudio T. Silva

VIS 2003

UNIVERSITY OF UTAH

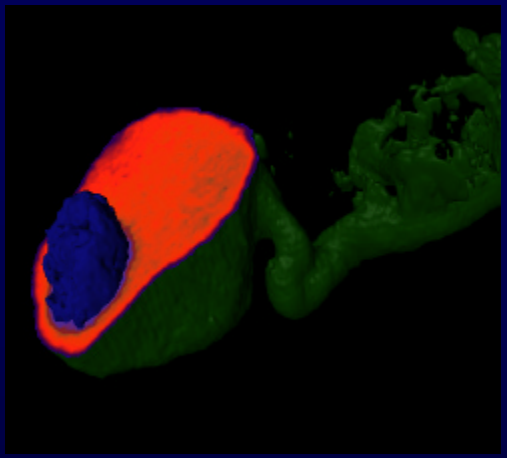
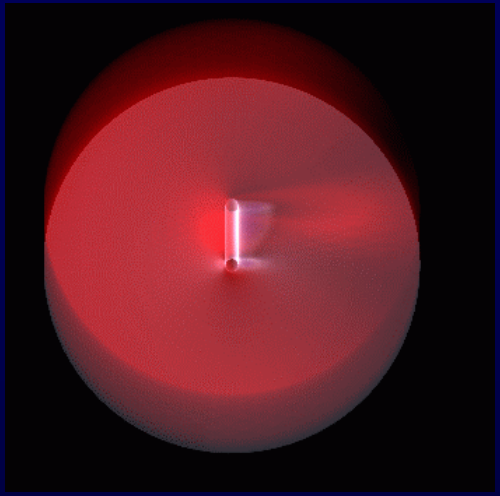
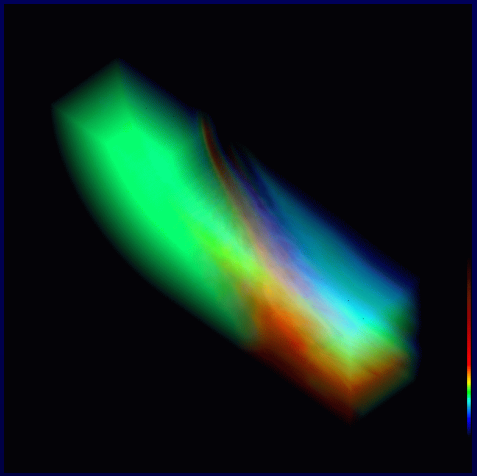
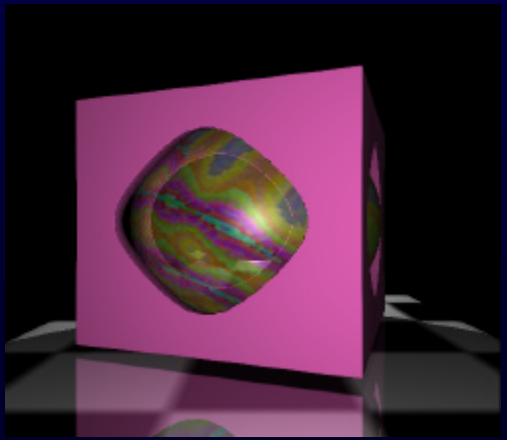
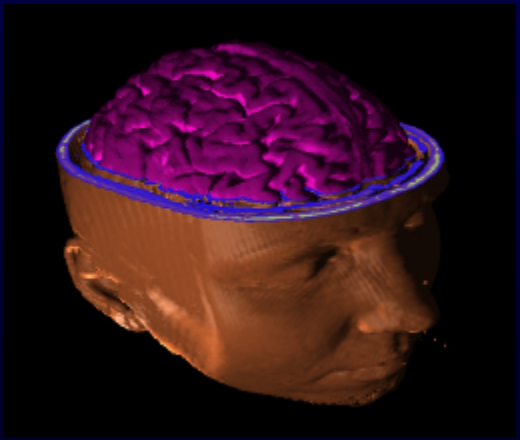
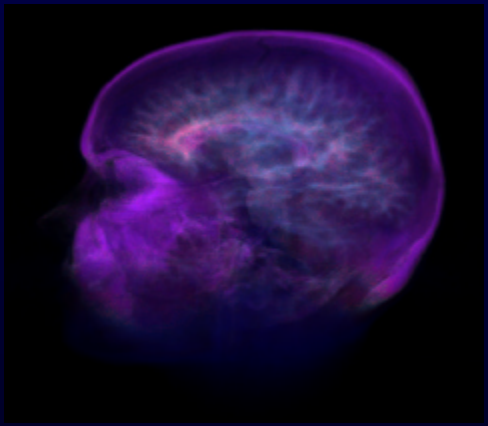
SCI INSTITUTE

THE UNIVERSITY OF UTAH

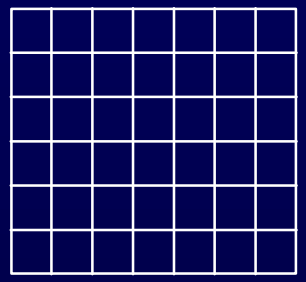
Outline

- Introduction to Volume Rendering
- Out-Of-Core Techniques
  - Introduction to External Memory Algorithms
  - Isosurface Generation
  - Direct Volume Rendering
- Hardware-Assisted Techniques
  - Introduction to Programmable Hardware
  - Direct Volume Rendering
  - Isosurface Generation (remarks)

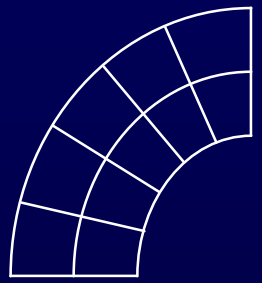
Introduction to Volume Rendering

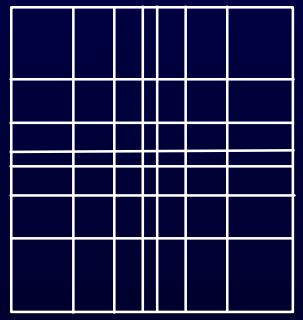
Grid Types



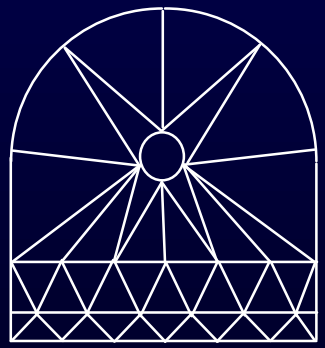
Regular



Curvilinear



Rectilinear

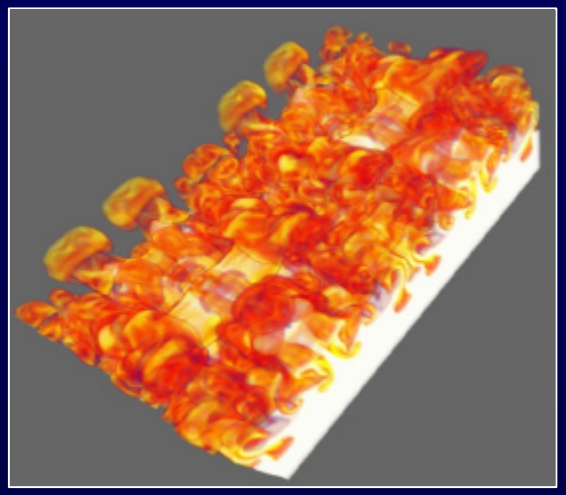


Irregular

Volume Rendering vs Isosurfaces

Direct Volume Rendering

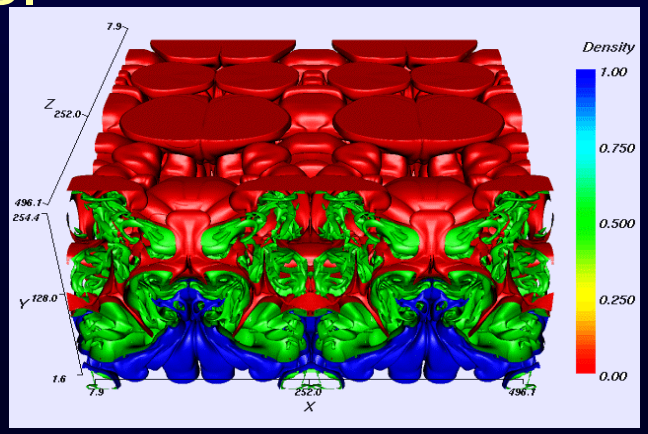
- Volume data → Image
- Looks “inside” the data



McPherson, LANL

Isosurfaces

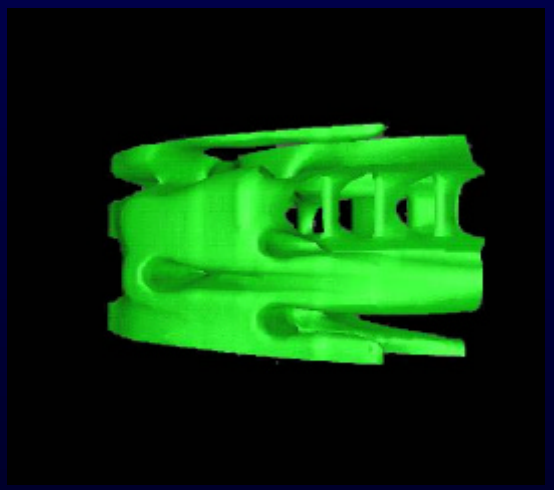
- Volume data → Polygon model
- Slice of the data

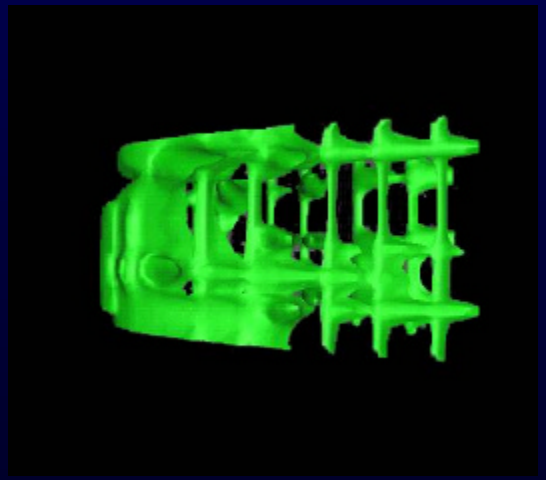


McPherson, LANL

Isosurfaces

For a query value q, find and display the isosurface of q: $C(q) = \{p \mid F(p) = q\}$





Isosurface Extraction Pipeline

Search Phase
O(N)

Generation Phase
O(K), K = N^(2/3)

Volume Data —**Marching Cubes**→ **Triangles**

↓ **simplification**

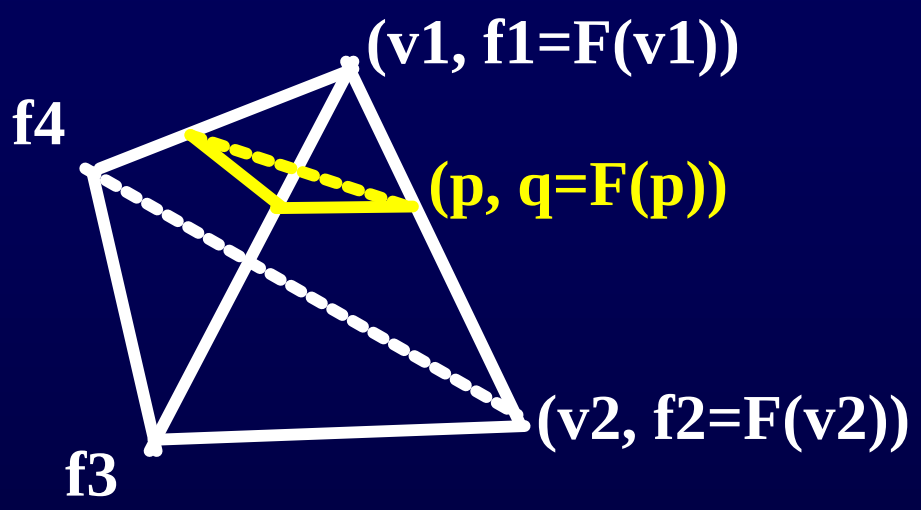
Triangles —**stripping**→ **Triangle Strips**

↓

Display

Search Phase: N cells → **K active cells**

Scalar Intervals and Active Cells



f_min = min(f1, f2, f3, f4)

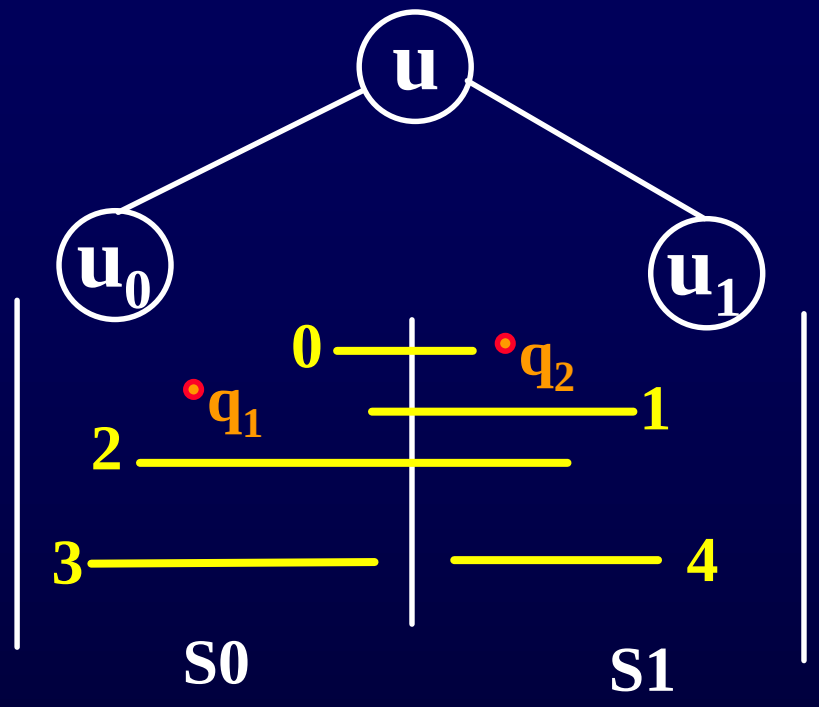
f_max = max(f1, f2, f3, f4)

Produce Interval I = [f_min, f_max] **for each cell**

Active cells
= **cells intersected by the isosurface**
= **cells with** f_min <= isovalue q <= f_max

[Cignoni et al 96]

Interval Tree [Edelsbrunner 81]



Intervals: crossing slab boundary -- u, else -- child
left list: left(u) = (2, 0, 1)
right list: right(u) = (1, 2, 0)

Isosurface Acceleration

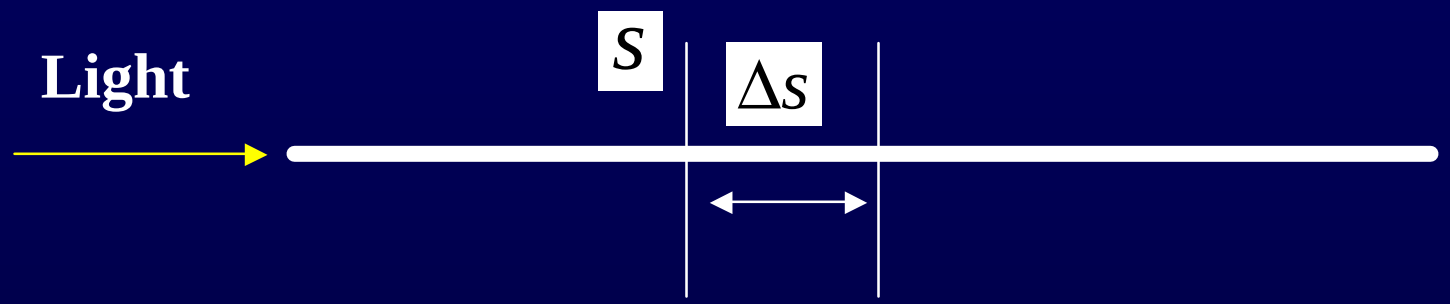
Span Space [Livnat et al]

Seed Cells [Bajaj et al, and others]

Optimal Contour Trees [Chiang et al, Carr et al]

Lots of other work…

Direct Volume Rendering: Optical Models



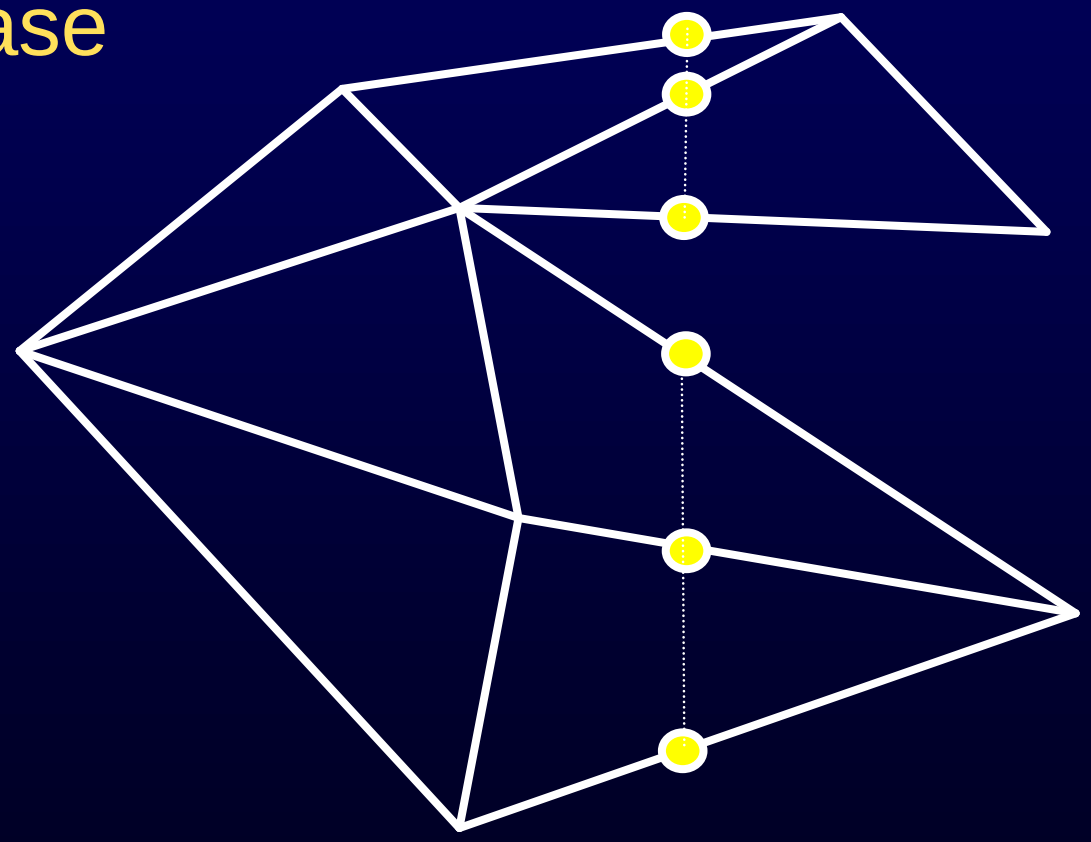
$$I(s+\Delta s) = I(s) + g(s)\Delta s - I(s)\Omega(s)\Delta s$$

$$I(s) = \int_0^s g(x) e^{-\int_0^x \Omega(y)dy} dx$$

Volume Rendering at a High Level

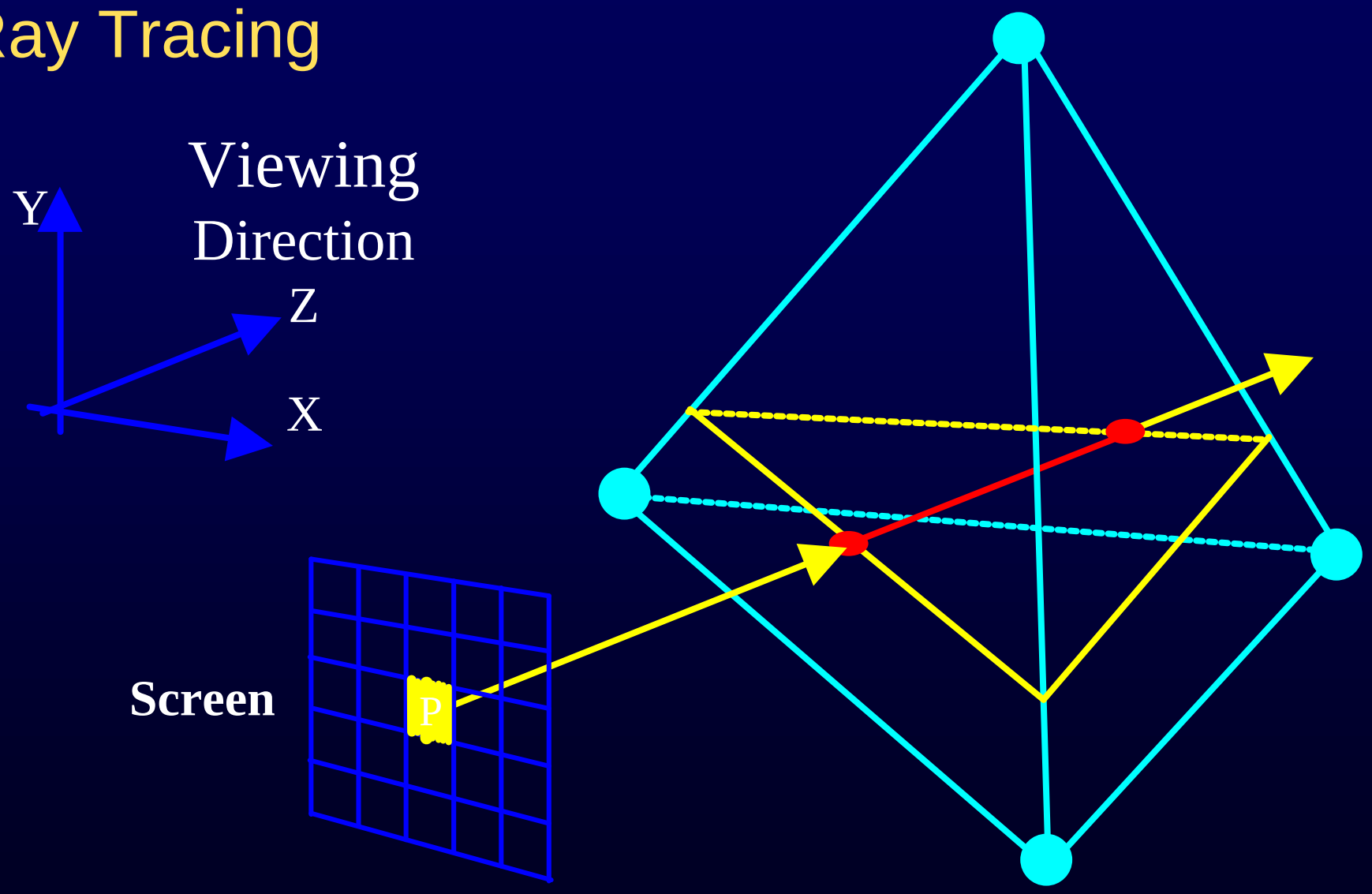
(a) Sampling Phase

(b) Sorting Phase



Unstructured Grid Volume Rendering

Ray Tracing



Outline

- Introduction to Volume Rendering
- Out-Of-Core Techniques
  - Introduction to External Memory Algorithms
  - Isosurface Generation
  - Direct Volume Rendering
- Hardware-Assisted Techniques
  - Introduction to Programmable Hardware
  - Direct Volume Rendering
  - Isosurface Generation (remarks)

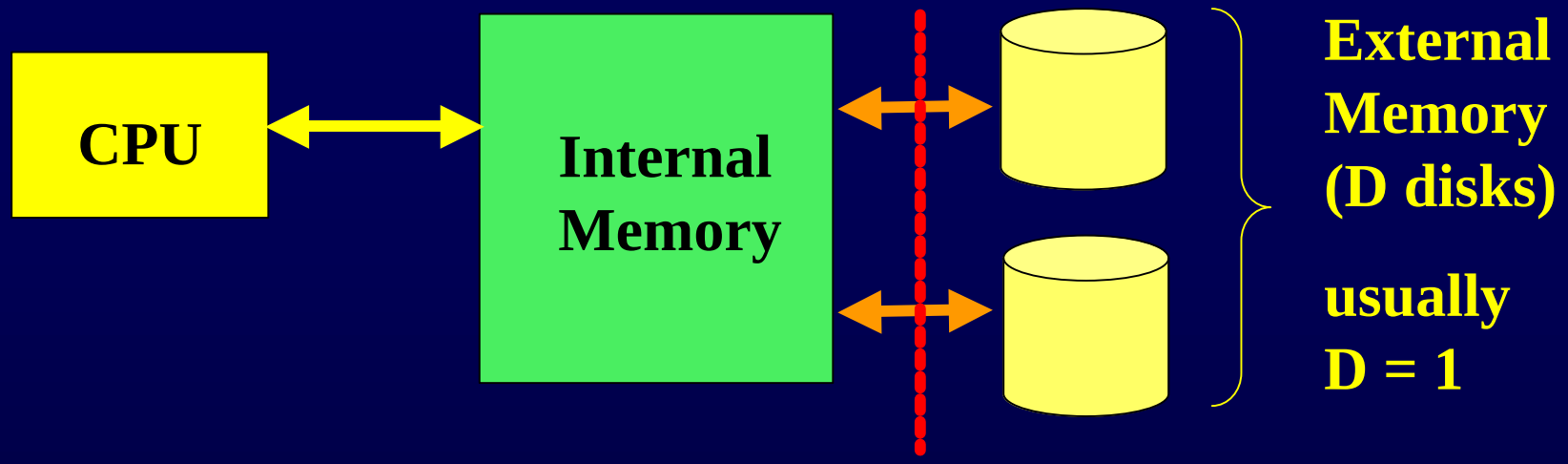
Principles of External Memory Algorithms

I/O Computational Model

Algorithmic Techniques

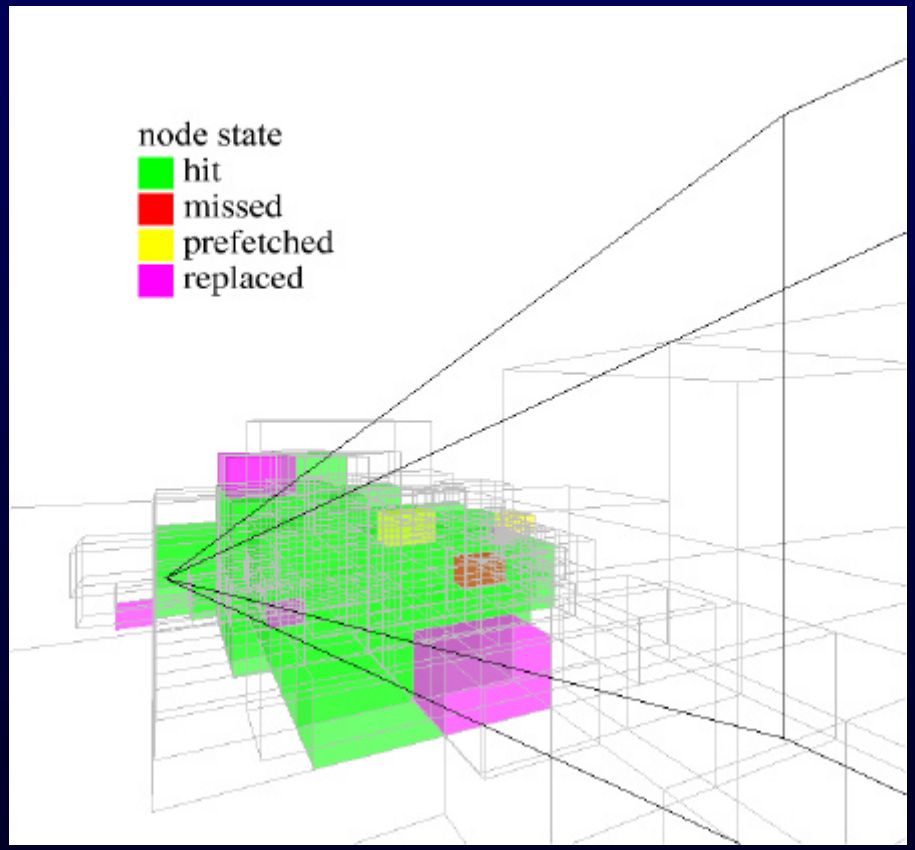
- Caching & Prefetching
  - [Cox-Ellsworth 1997; Sutton-Hansen 2000; Bajaj et al 1999; etc]
- External Merge Sort
- Out-Of-Core Pointer De-Referencing
- Meta-Cell Technique [Chiang and Silva 1998]
- Indexing (B-Tree-Like Data Structures)

I/O Computational Model



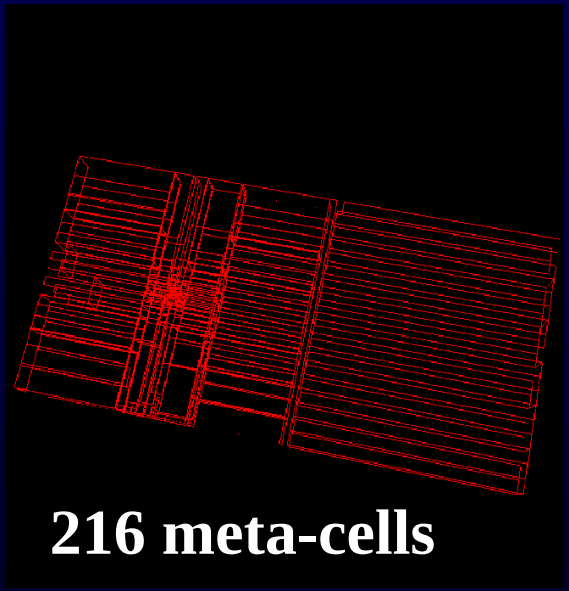
- B: # items fitting in one disk block
- One I/O reads/writes one block (B items)
- Virtual memory vs Out-of-core techniques
- Batched computations (e.g. external sorting)
- On-line computations (e.g. B-tree)

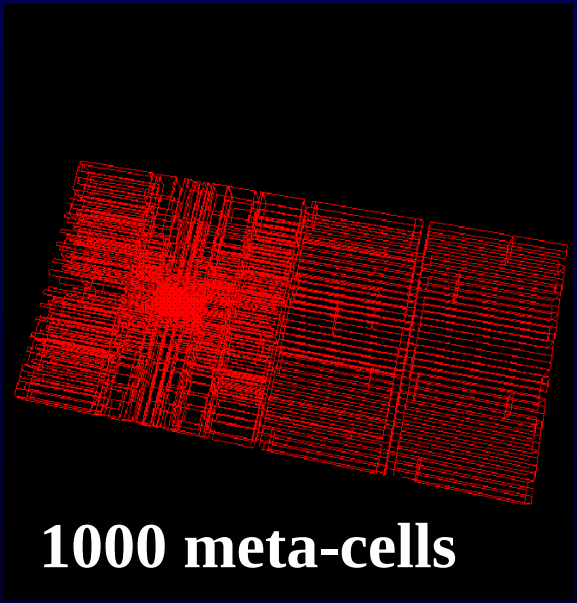
Caching & Prefetching



Meta-Cell Technique [Chiang and Silva 1998]

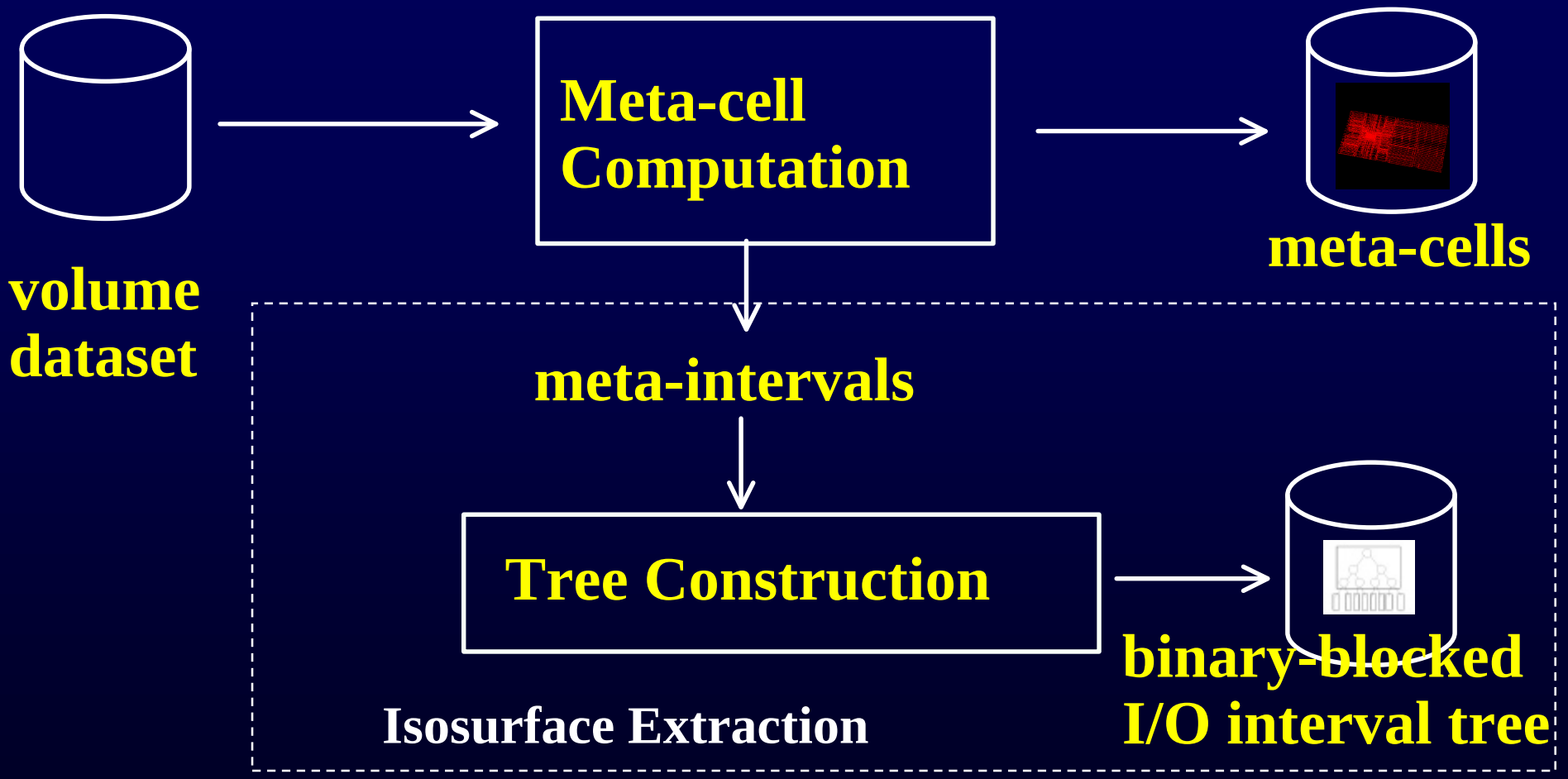
Out-of-core technique for creating spatially coherence geometry for <u>unstructured grids</u>



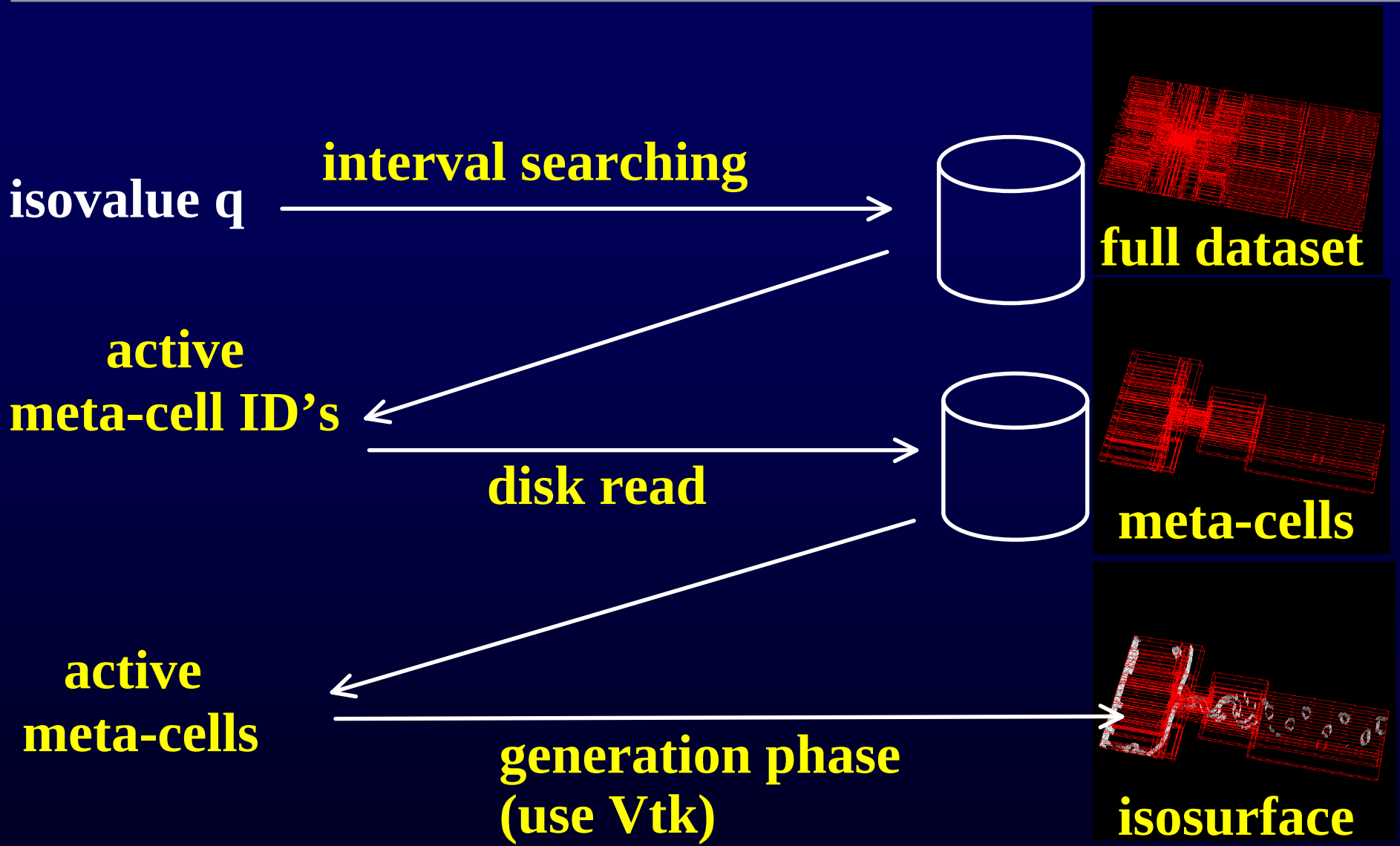


Out-Of-Core Isosurface Extraction

Preprocessing Pipeline



Isosurface Query Pipeline



Out-Of-Core Direct Volume Rendering

- Sampling Phase does not change
- The part that changes is the <u>sorting phase</u>

Sorting

Application

Object Space

Rasterization

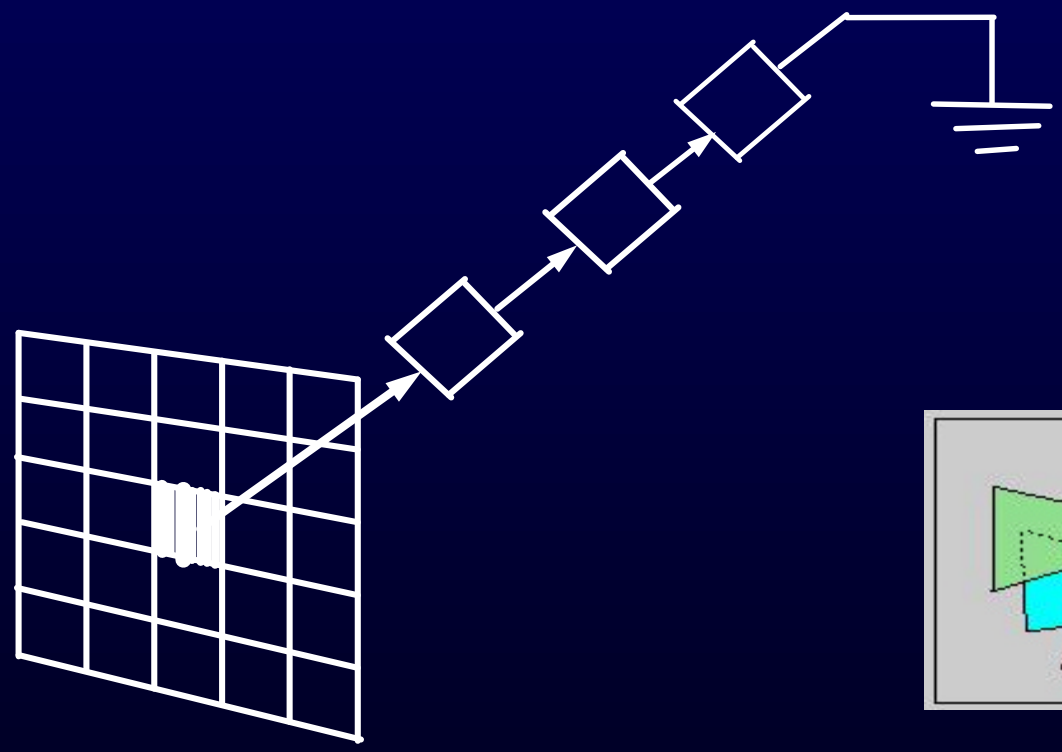
Image-Space Sorting

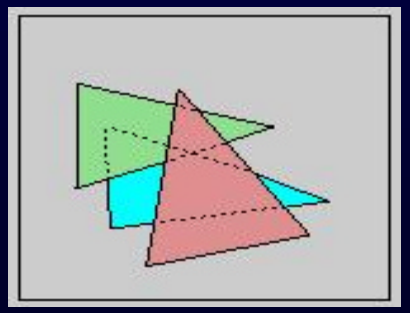
Display

i.e., let's sort the pixes!

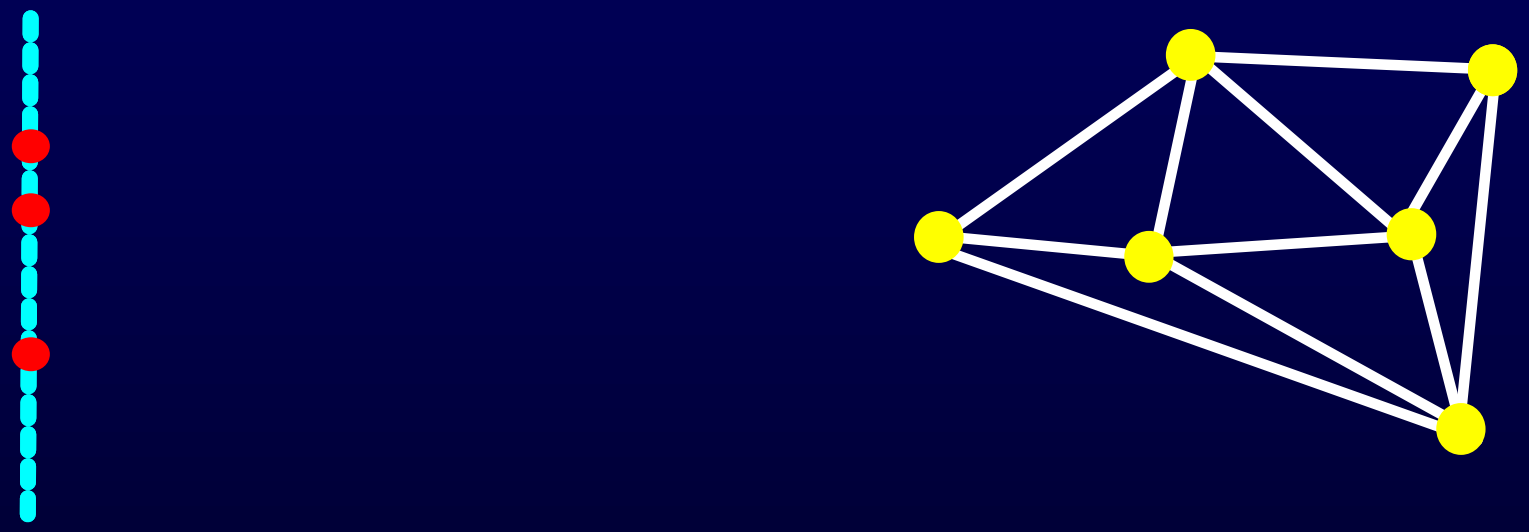
Image-Order Sorting: Carpenter's A-Buffer

Idea: Keep a list of intersections for each pixel

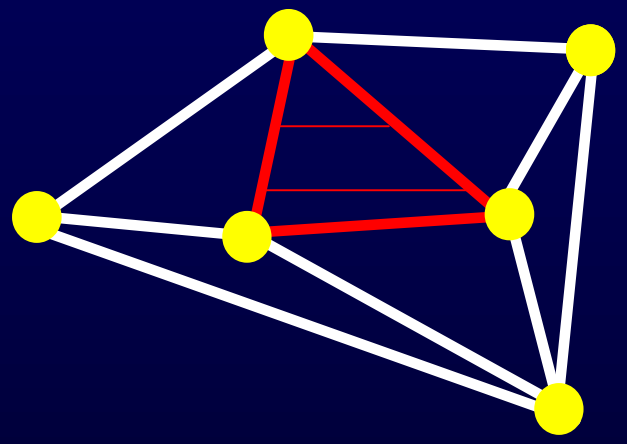




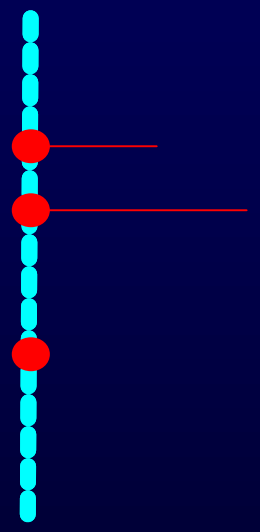
Cell-Projection With An A-Buffer

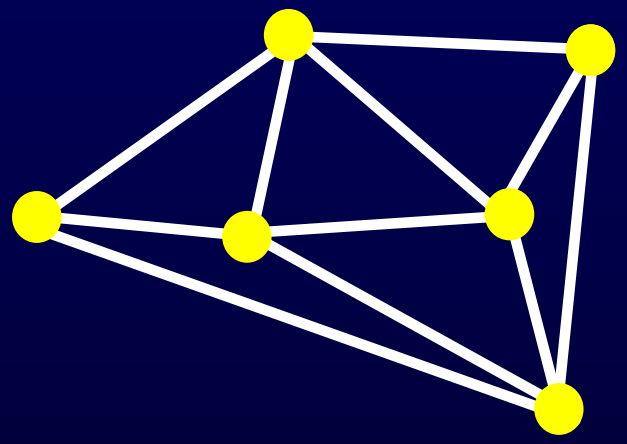


Cell-Projection With An A-Buffer

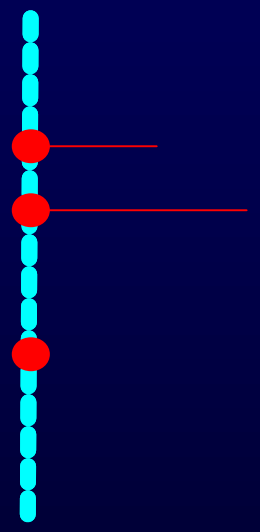


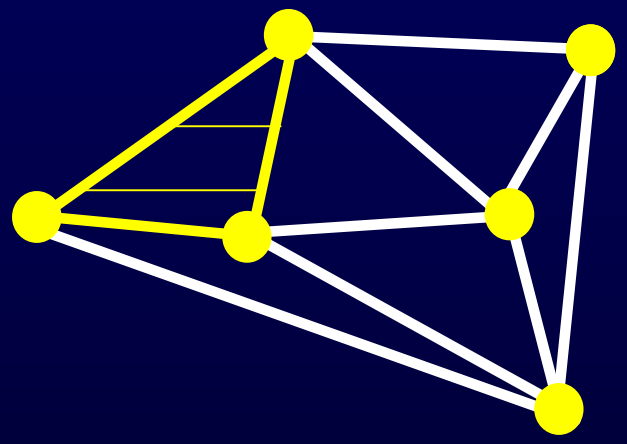
Cell-Projection With An A-Buffer



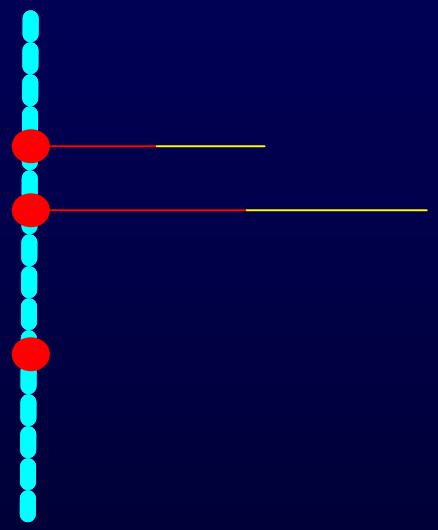


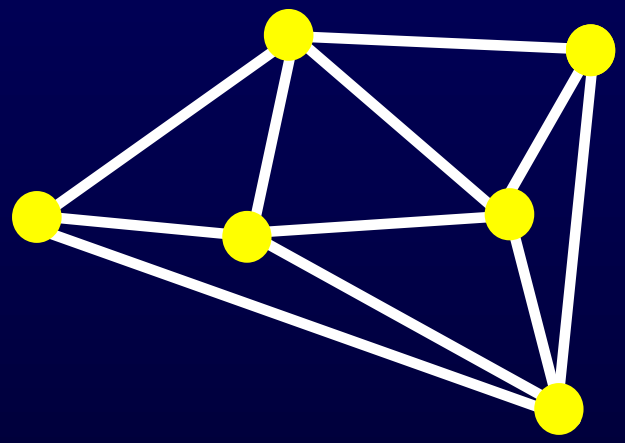
Cell-Projection With An A-Buffer



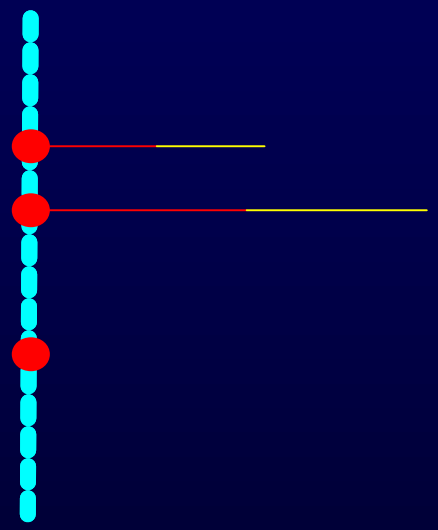


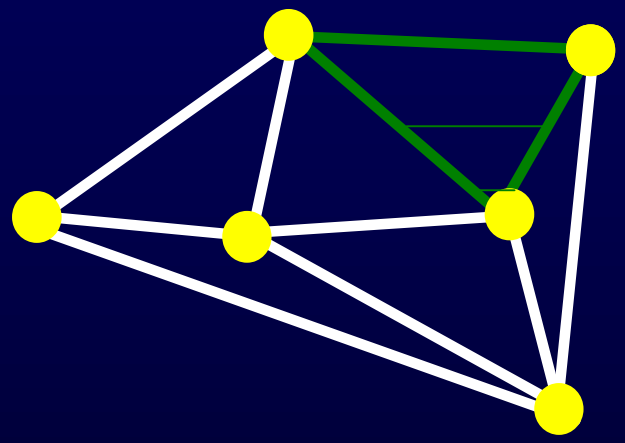
Cell-Projection With An A-Buffer



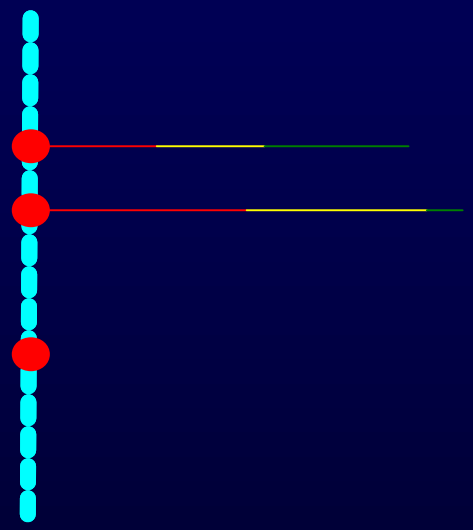


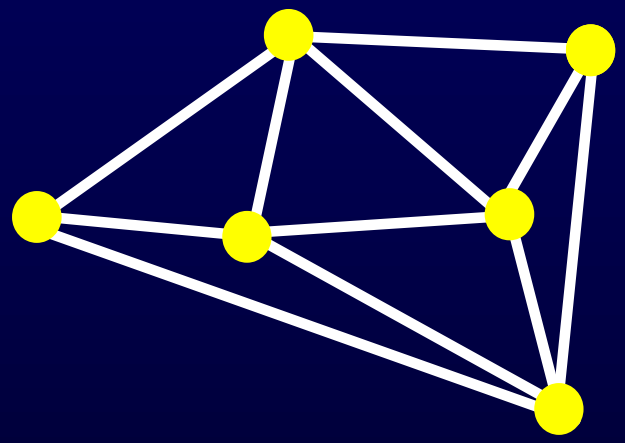
Cell-Projection With An A-Buffer



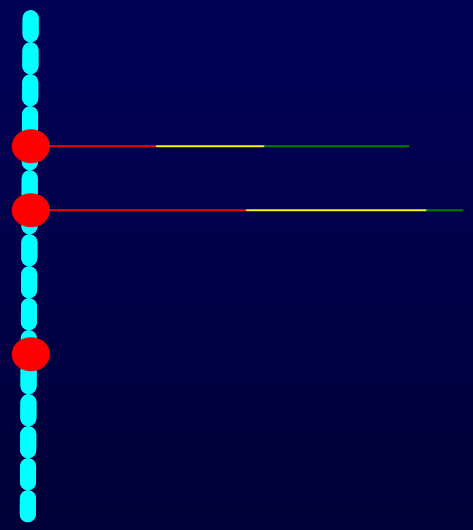


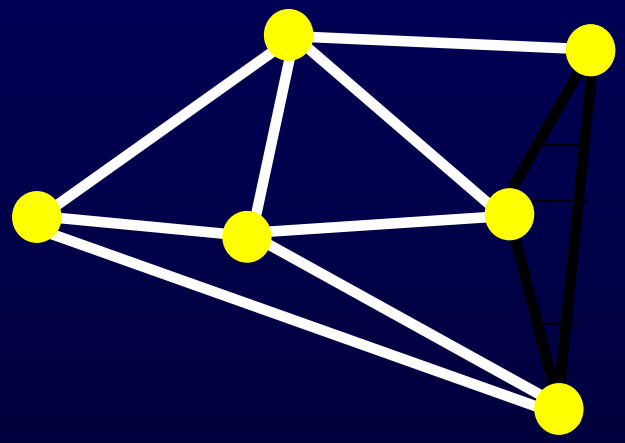
Cell-Projection With An A-Buffer



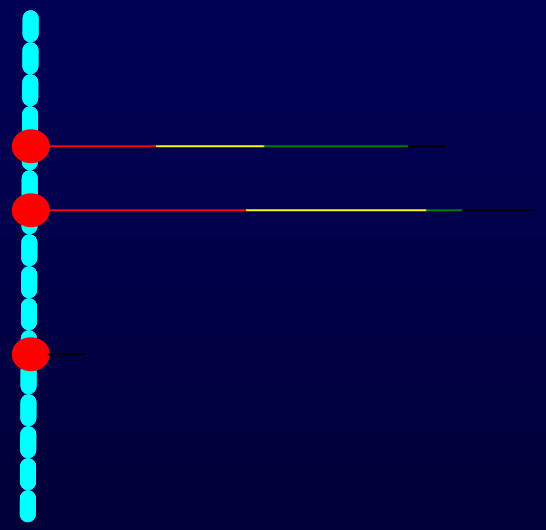


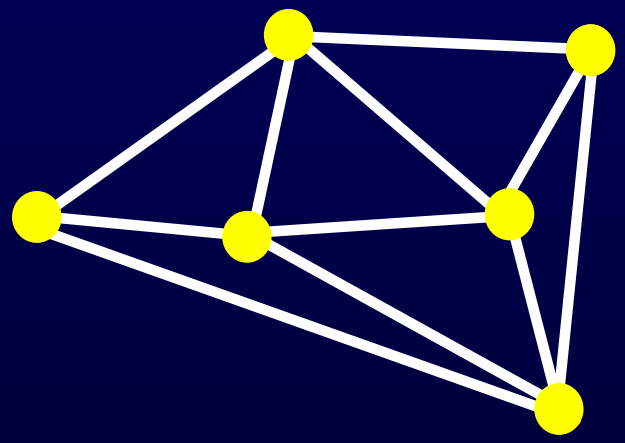
Cell-Projection With An A-Buffer



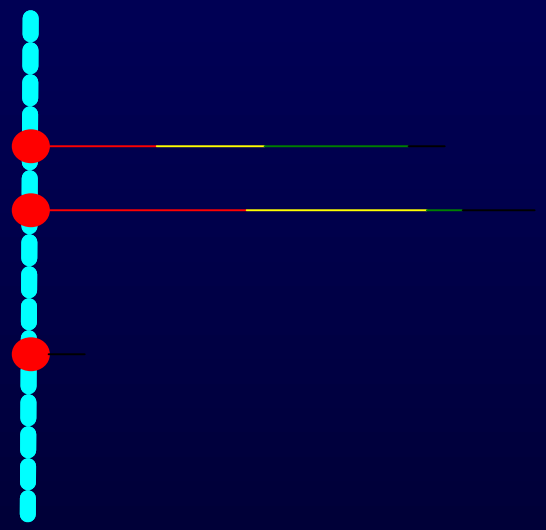


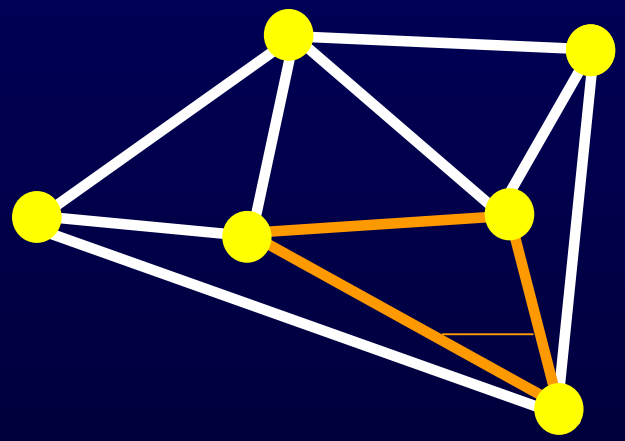
Cell-Projection With An A-Buffer



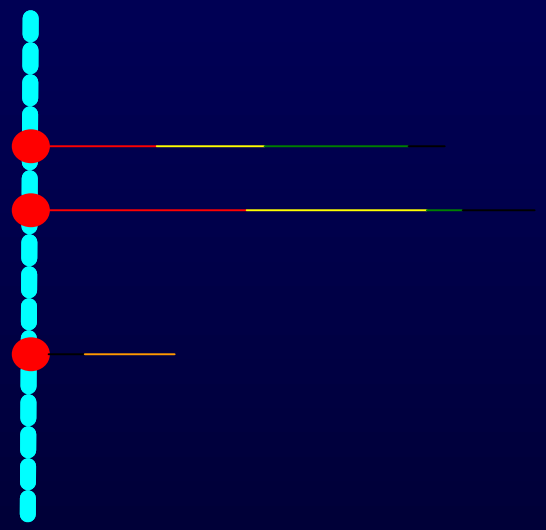


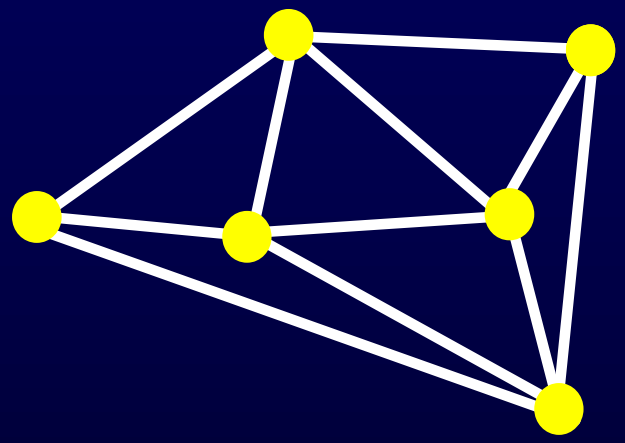
Cell-Projection With An A-Buffer



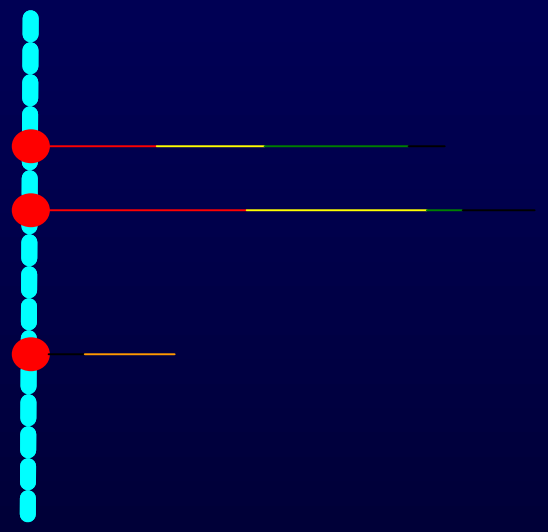


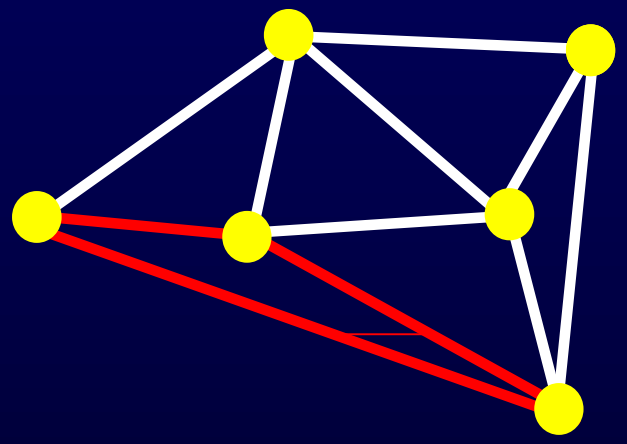
Cell-Projection With An A-Buffer



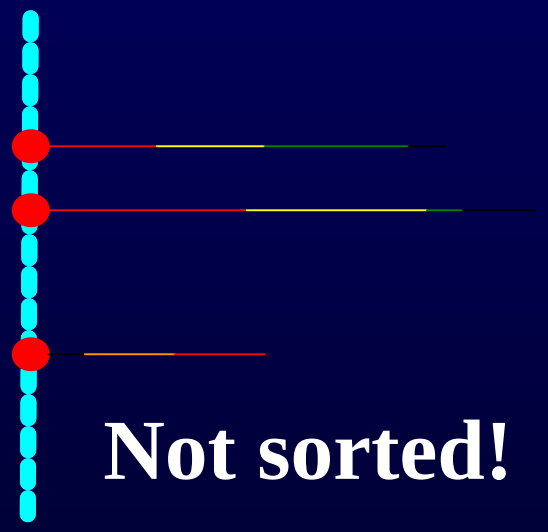


Cell-Projection With An A-Buffer

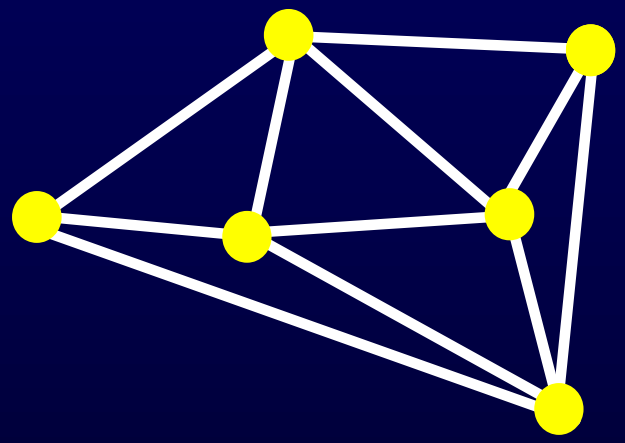




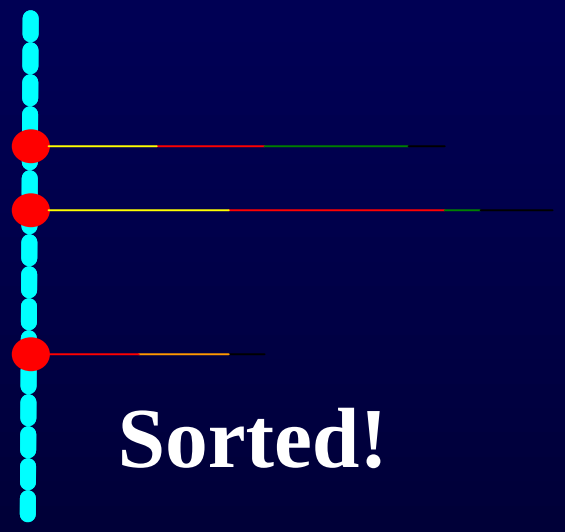
Cell-Projection With An A-Buffer

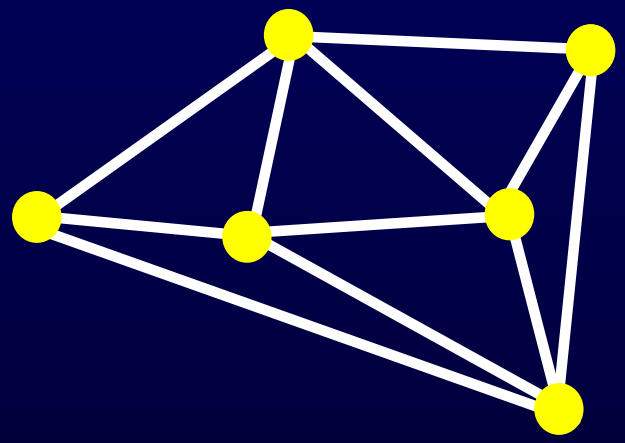


Not sorted!

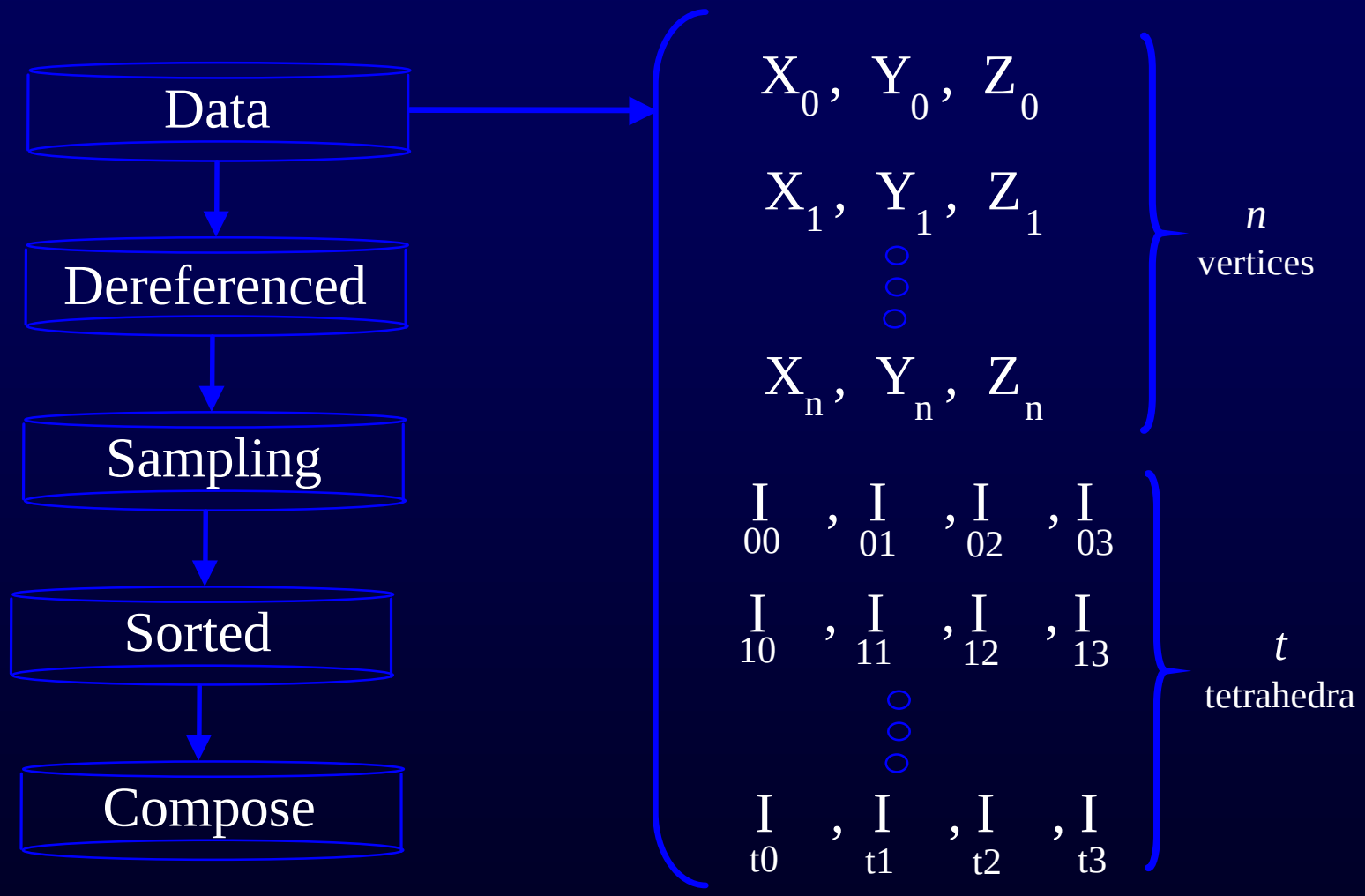


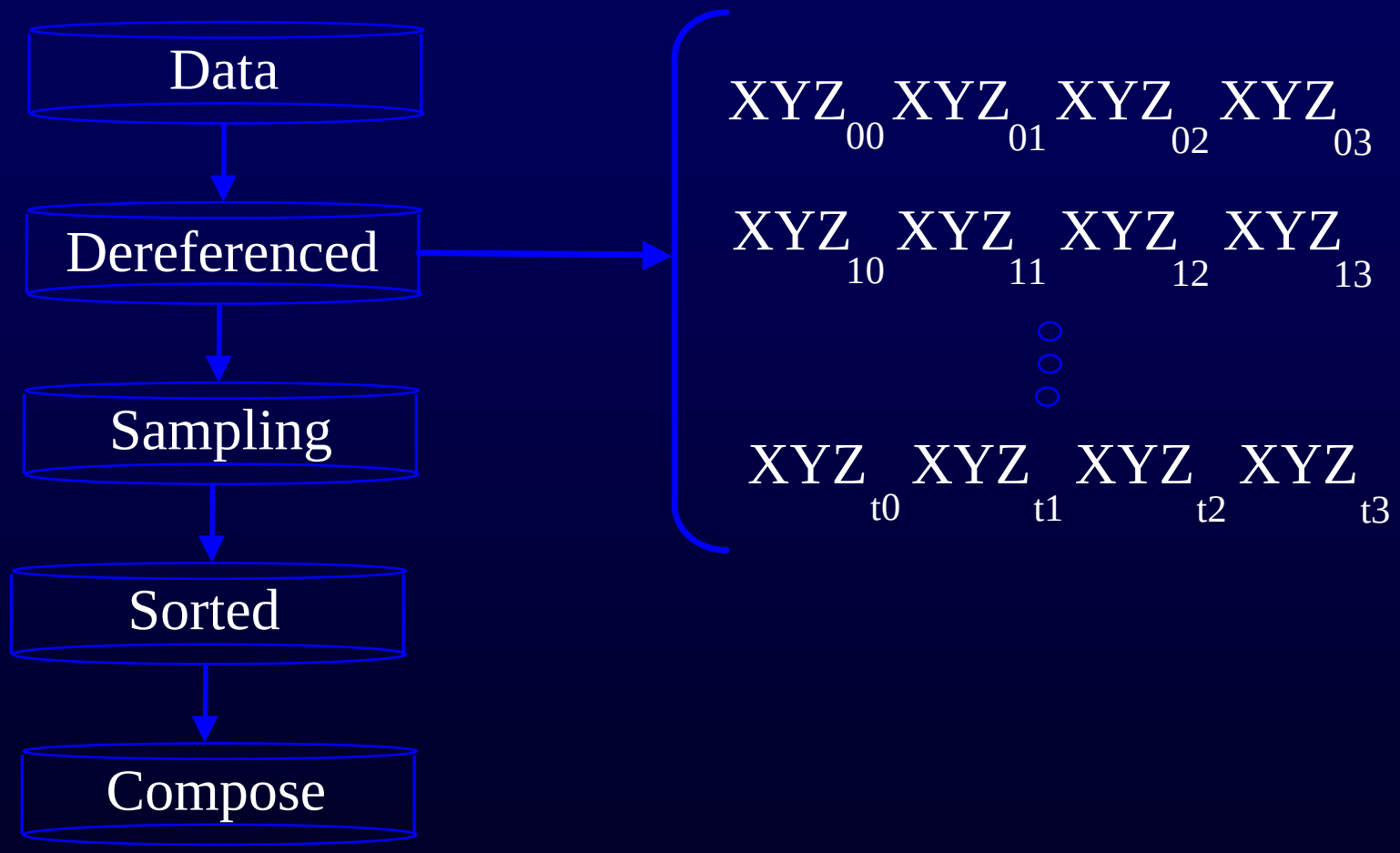
Cell-Projection With An A-Buffer

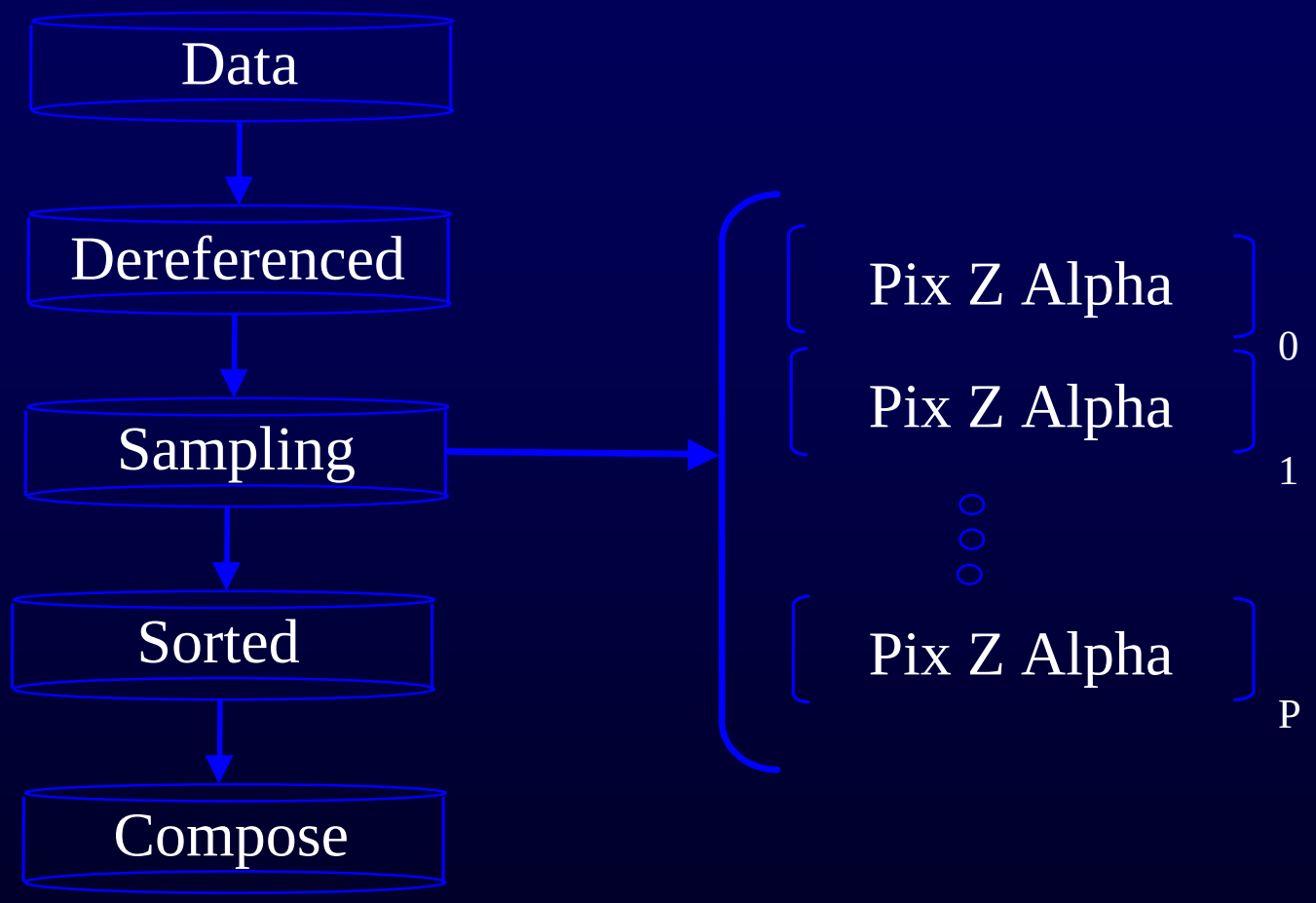


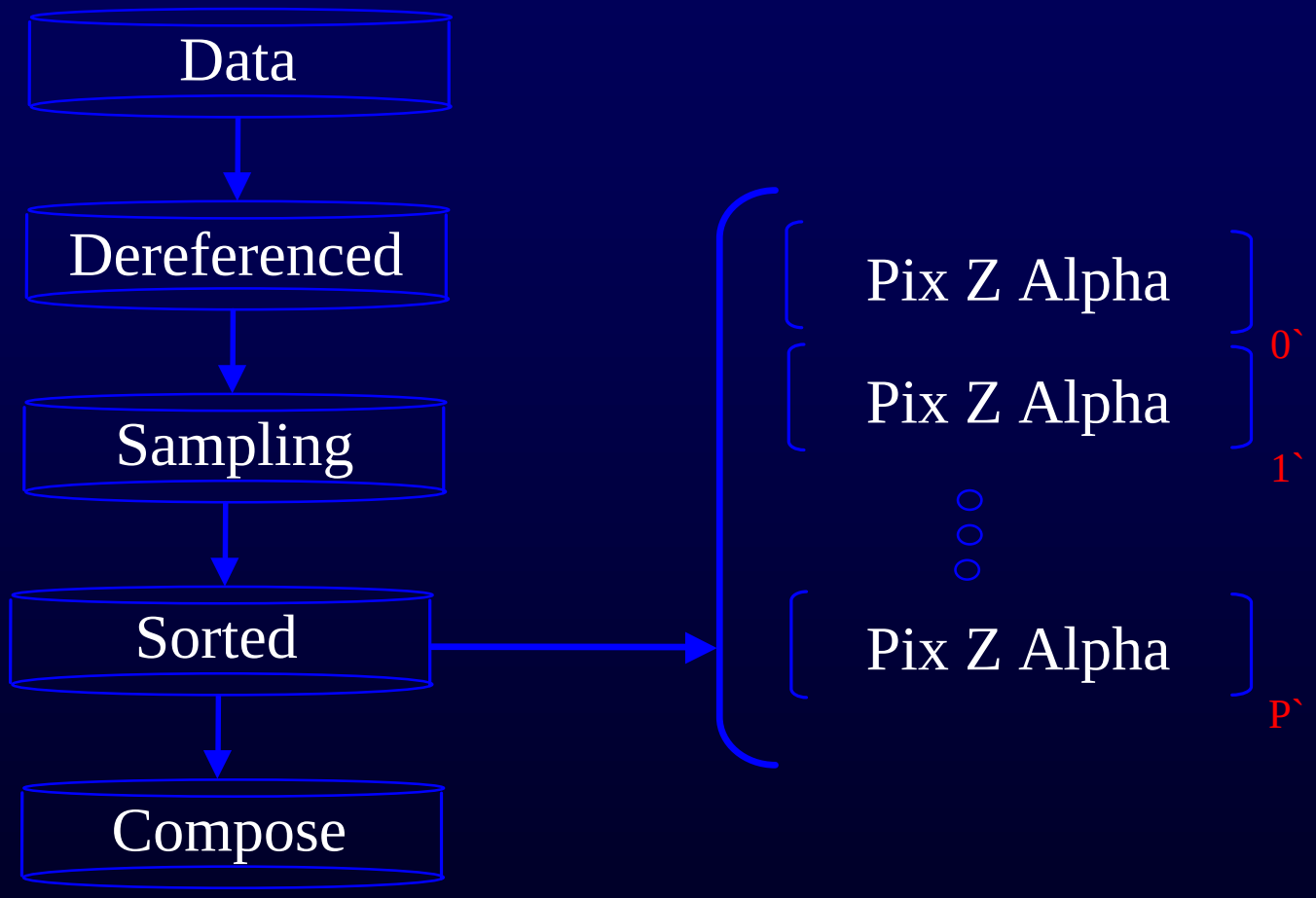


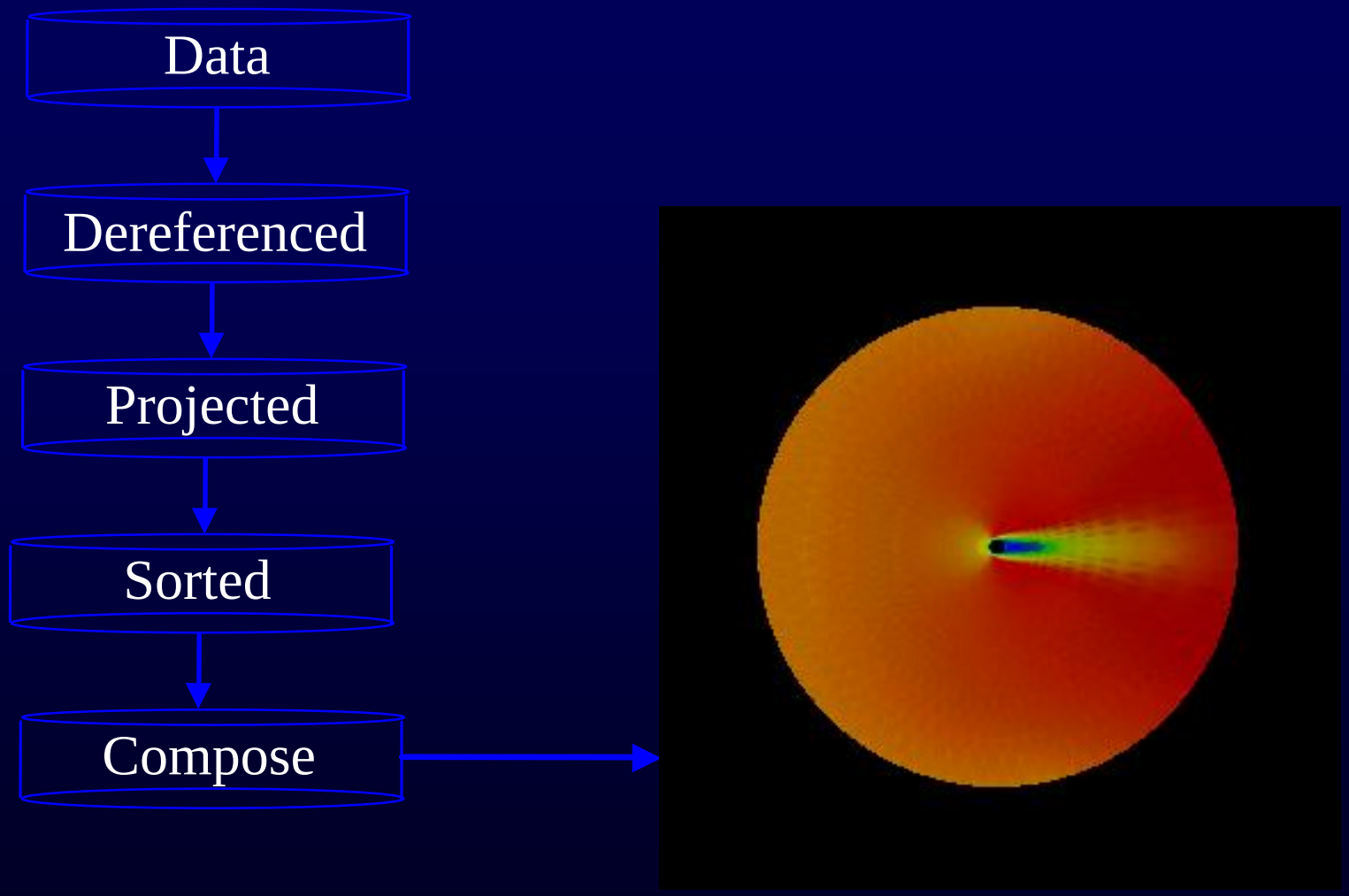
A Simple Memory-Insensitive Technique




Data
Dereferenced
Sampling
Sorted
Compose
XYZ_{00} XYZ_{01} XYZ_{02} XYZ_{03}
XYZ_{10} XYZ_{11} XYZ_{12} XYZ_{13}
XYZ_{t0} XYZ_{t1} XYZ_{t2} XYZ_{t3}


Data
Dereferenced
Sampling
Sorted
Compose
Pix Z Alpha
0
Pix Z Alpha
1
Pix Z Alpha
P

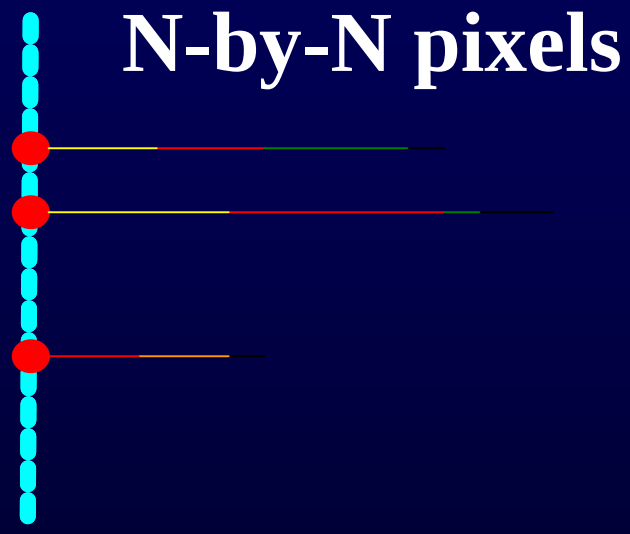

Data
Dereferenced
Sampling
Sorted
Compose
Pix Z Alpha
0`
Pix Z Alpha
1`
Pix Z Alpha
P`

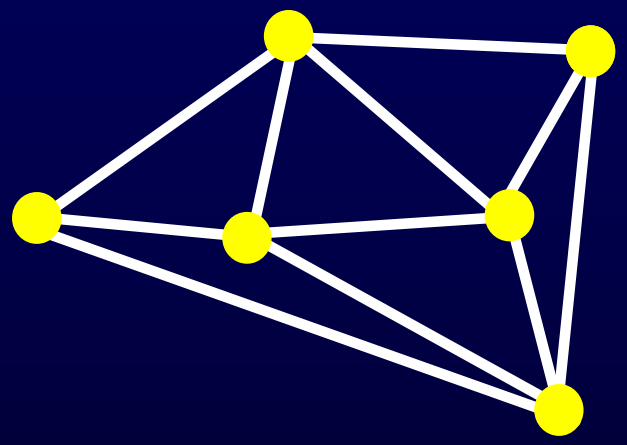

Data
Dereferenced
Projected
Sorted
Compose

Features of Memory-Insensitive Algorithm

- Dataset can be arbitrarily large
- Easily to implement
  - Cell intersection code
  - Sample Integration code
  - External Memory sort
- Transfer function modifications for same viewpoint are fast! (also for time-varying data)

Complexity Analysis





Number of Intersections: $O(c\ N^2)$

Problems:

1. **Time:** **sorting intersections takes too long!**
2. **Memory:** **storage too high!**

Sorting

Application

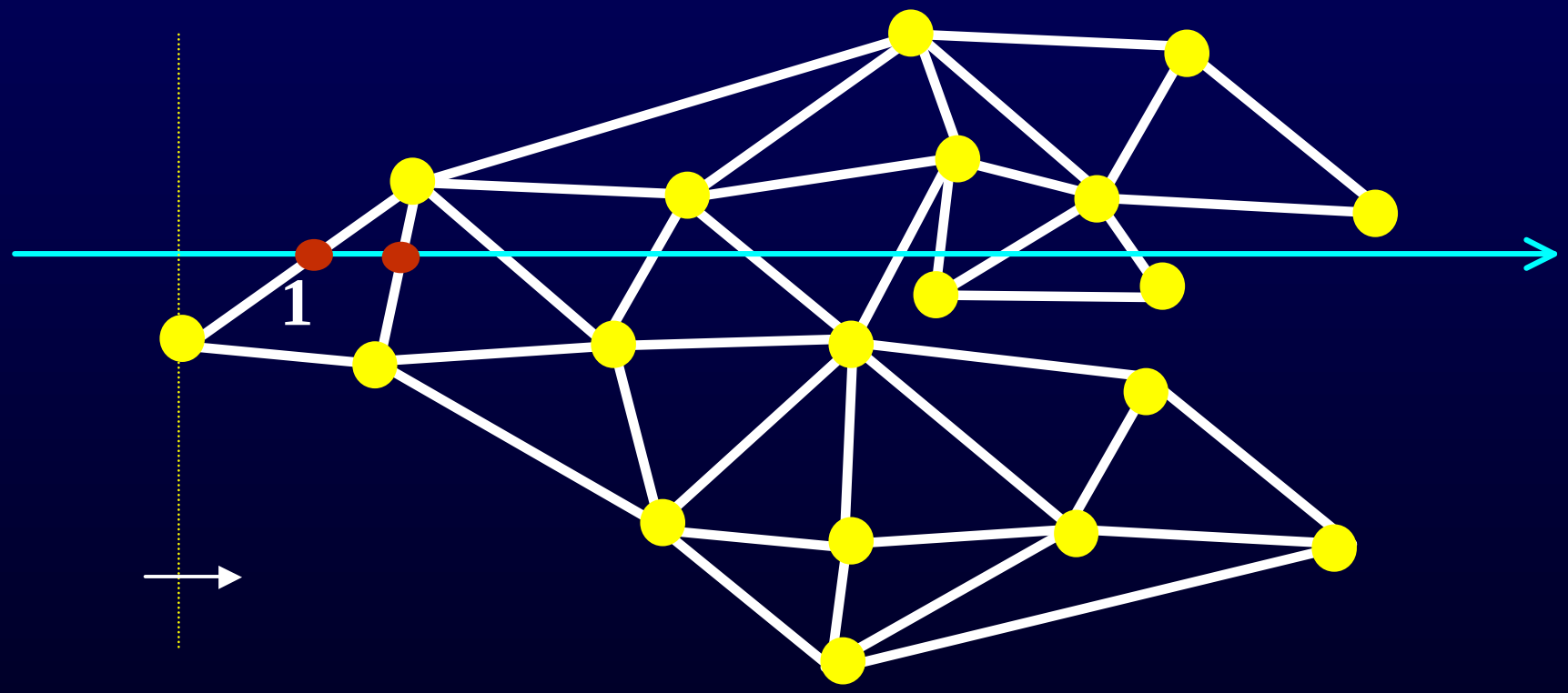
Object-Space Sorting

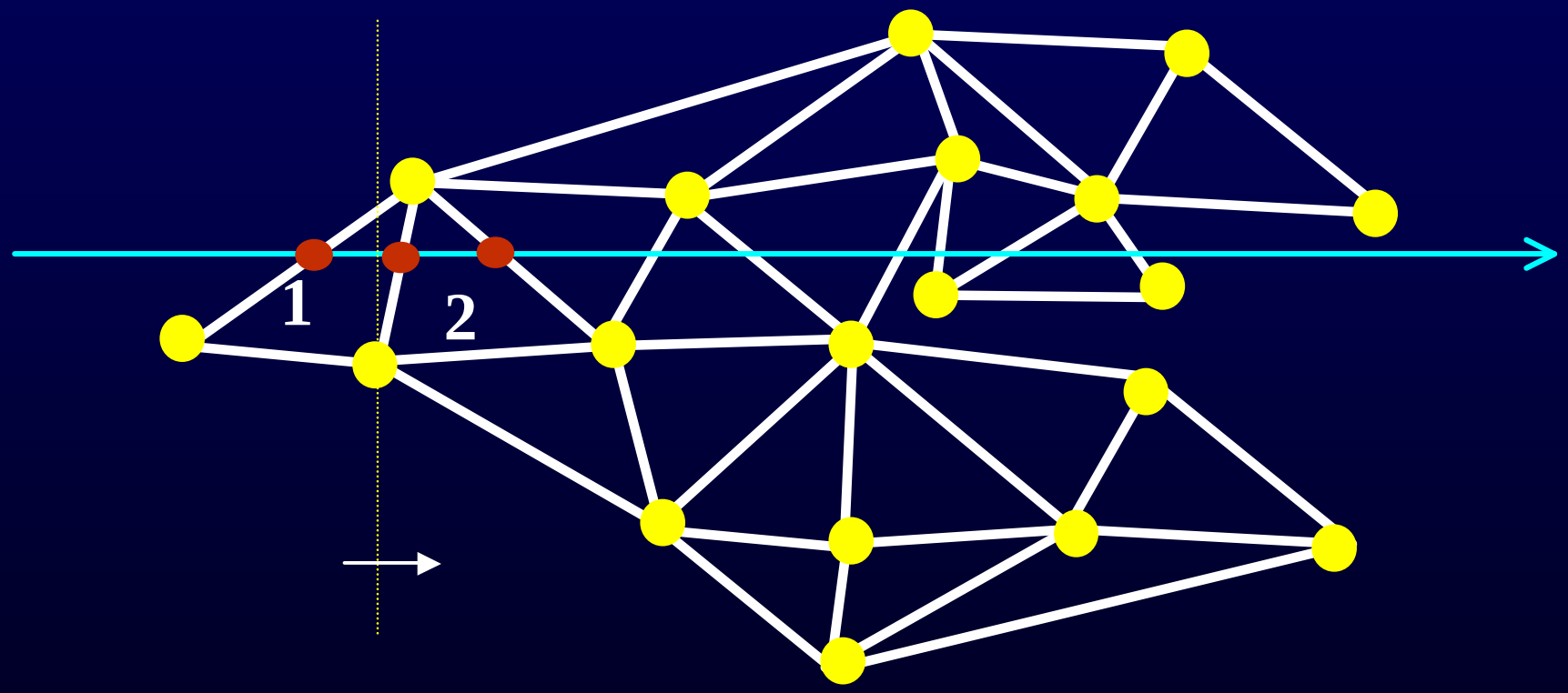
Rasterization

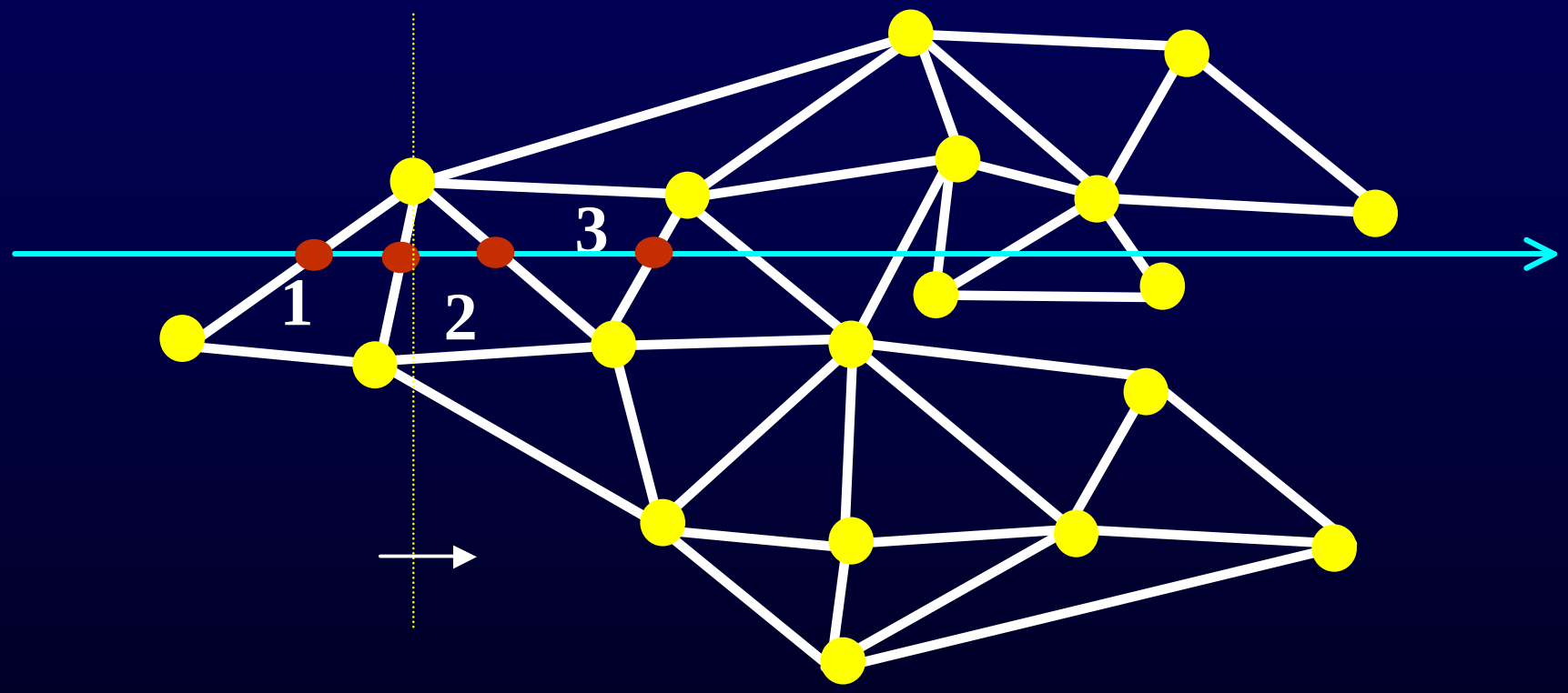
Image-Space Sorting

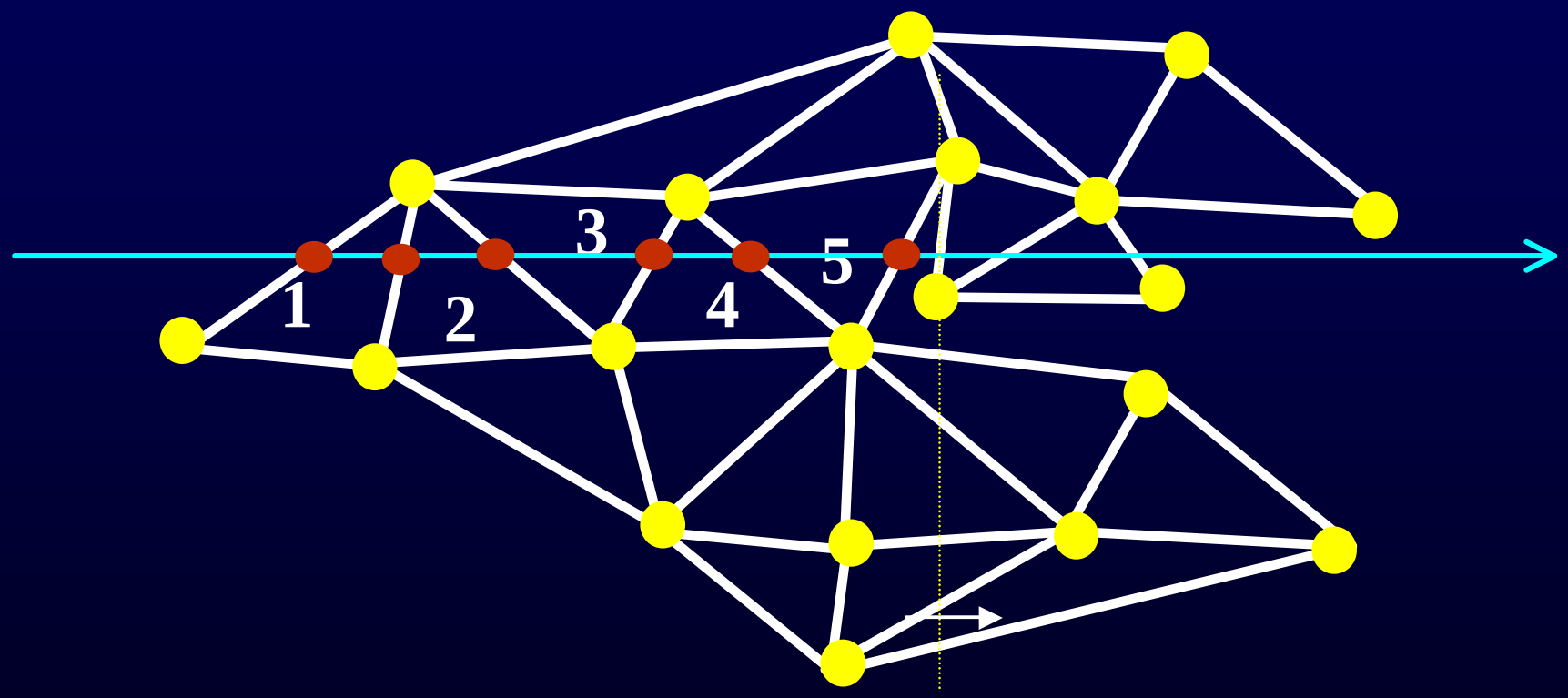
Display

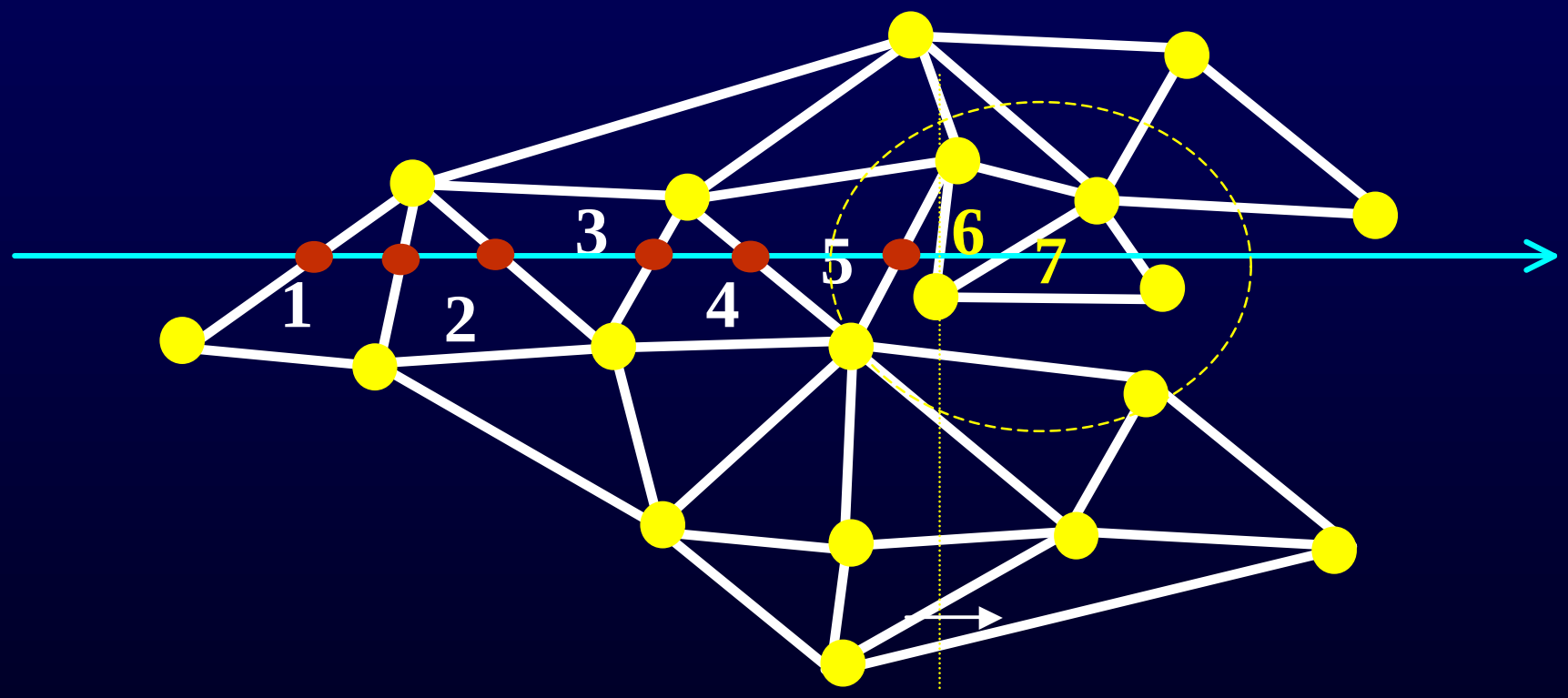
Approximate Object-Space Sorting

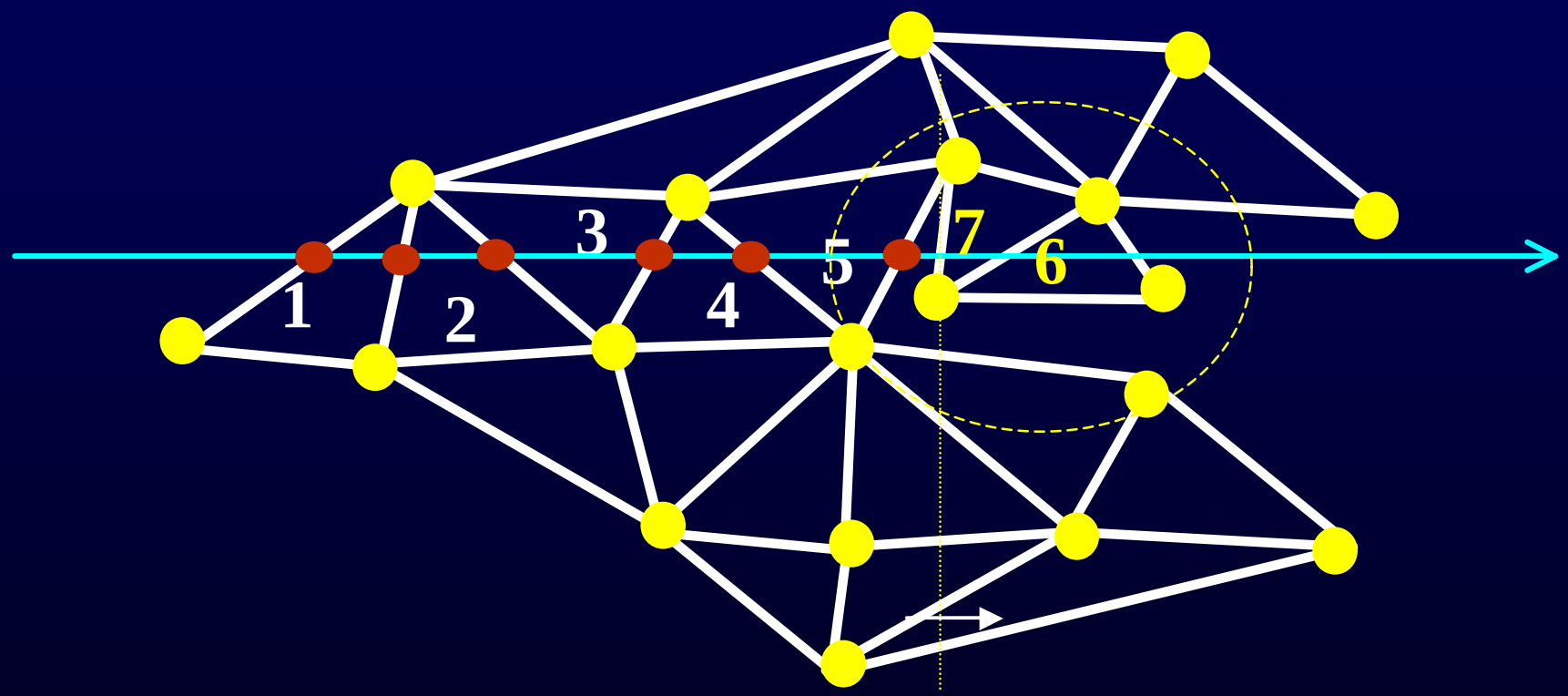



1
2

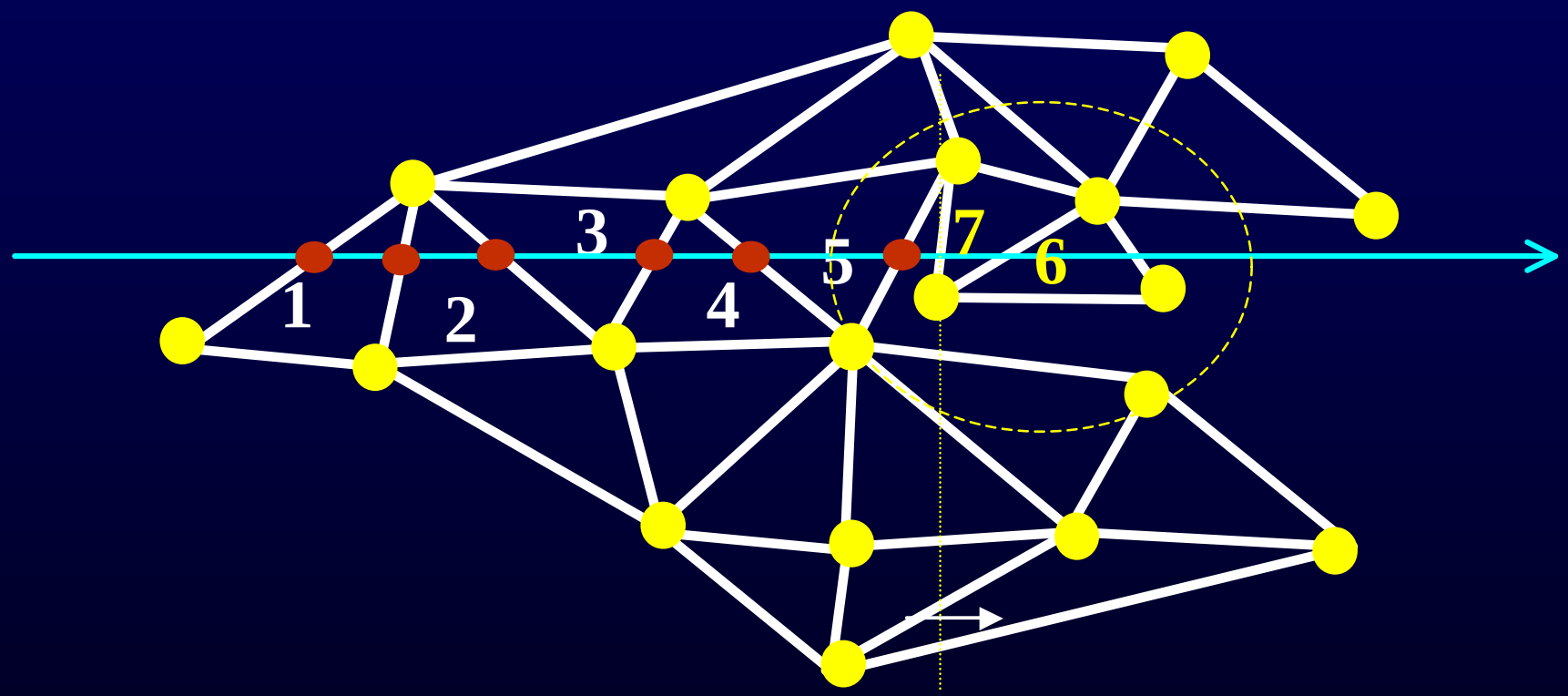

1
2
3


1
2
3
4
5


1
2
3
4
5
6
7



A Solution: Use an insertion-sort A-buffer!

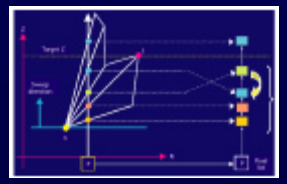


What about the space problem ?

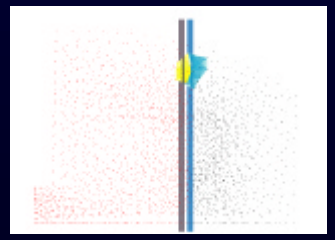
→ Use a conservative bound on the intersections

ZSWEEP: Sort in Image and Object Space

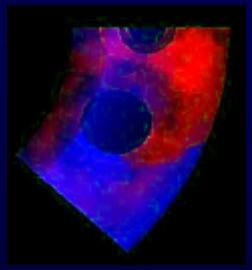
Do an approximate sorting of the cells (e.g., based on the order of the vertices), and use an insertion sort for the intersections – solves the sorting overhead



Propose a conservative technique for limiting the memory usage, by performing compositing on the intersections, anytime the lists become "too large" – solves the memory overhead

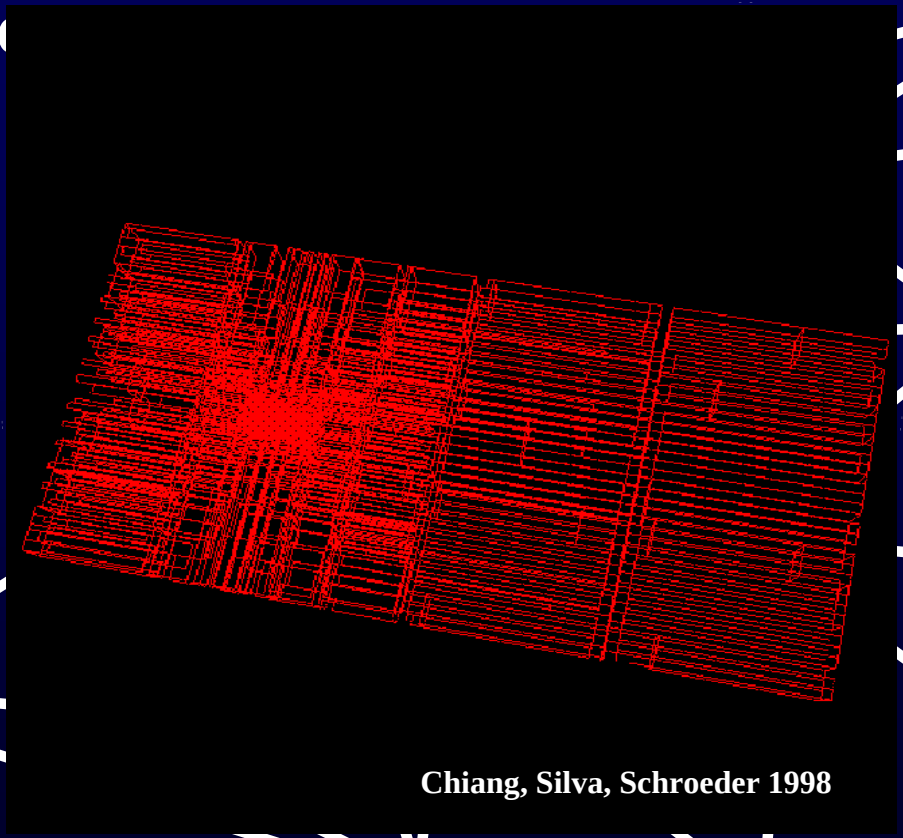


ZSWEEP - DEMO



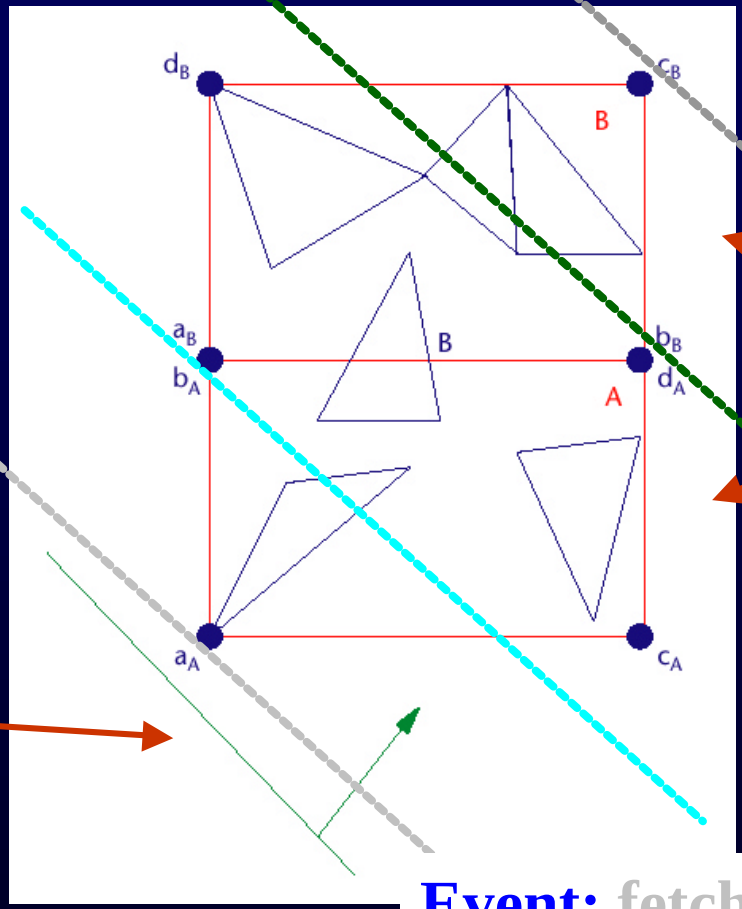
Blocking for Out-Of-Core Rendering

Meta-



Out-Of-Core ZSWEEP: Data Management

[Farias and Silva 2001a]



fetch B

dump B

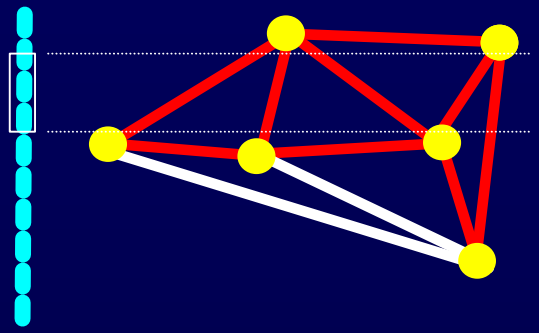
Meta-cel

dump A

Sweep Plane

Event: fetch A

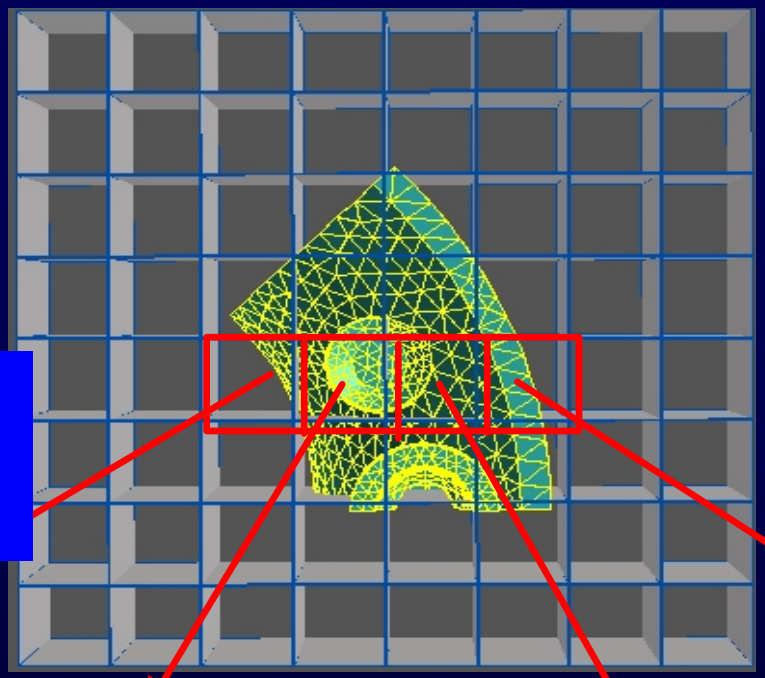
Tile-Based ZSWEEP

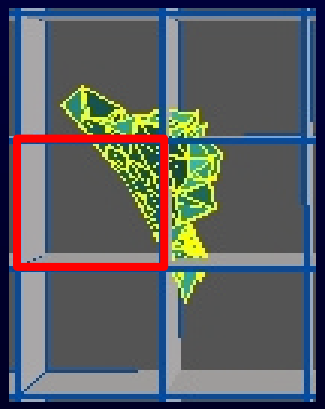


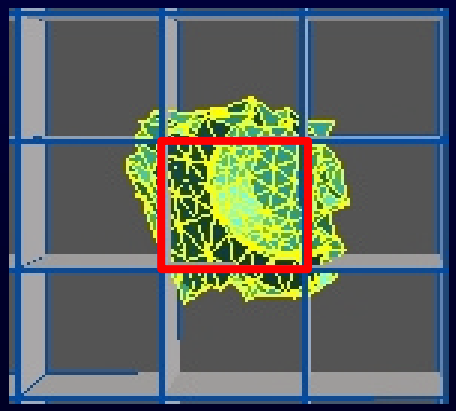
SGI R10K L2 – 2MB

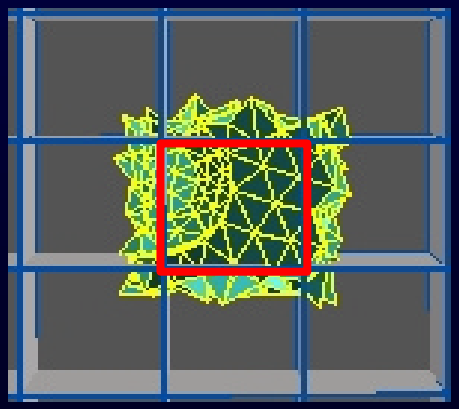
ZSWEEP: 56% hit rate

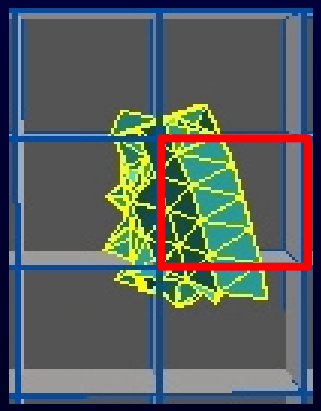
TB-ZSWEEP: 94% hit rate











Out-Of-Core ZSWEEP Results

Generating 2048x2048 Image (sec)				
	Original ZSWEEP		OOC-ZSWEEP	
	Memory	Time	Memory	Time
Blunt Fin	330	386	6.2	331
Combustion Chamber	330	407	6.2	423
Oxygen Post	350	775	13.2	667
Delta Wing	380	639	24.2	537

Video

QuickTime™ and a
Photo decompressor
are needed to see this picture.

Outline

- Introduction to Volume Rendering
- Out-Of-Core Techniques
  - Introduction to External Memory Algorithms
  - Isosurface Generation
  - Direct Volume Rendering
- Hardware-Assisted Techniques
  - Introduction to Programmable Hardware
  - Direct Volume Rendering
  - Isosurface Generation (remarks)

Hardware-Assisted Volume Rendering

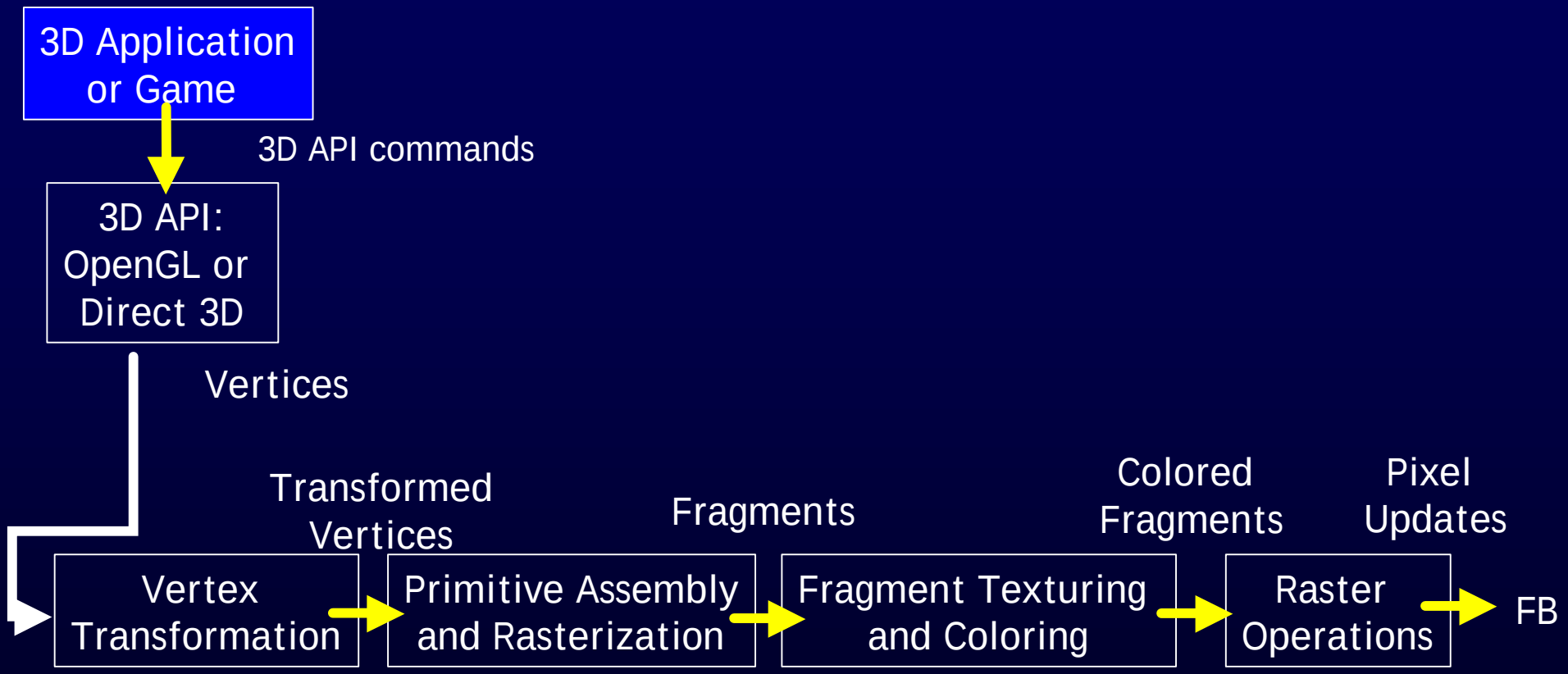
- Introduction to Programmable Hardware
- Texture-Based Volume Rendering
- Rendering Unstructured Grids

Introduction to Programmable Hardware

- Graphics Pipeline
- Programmable Graphics Pipeline

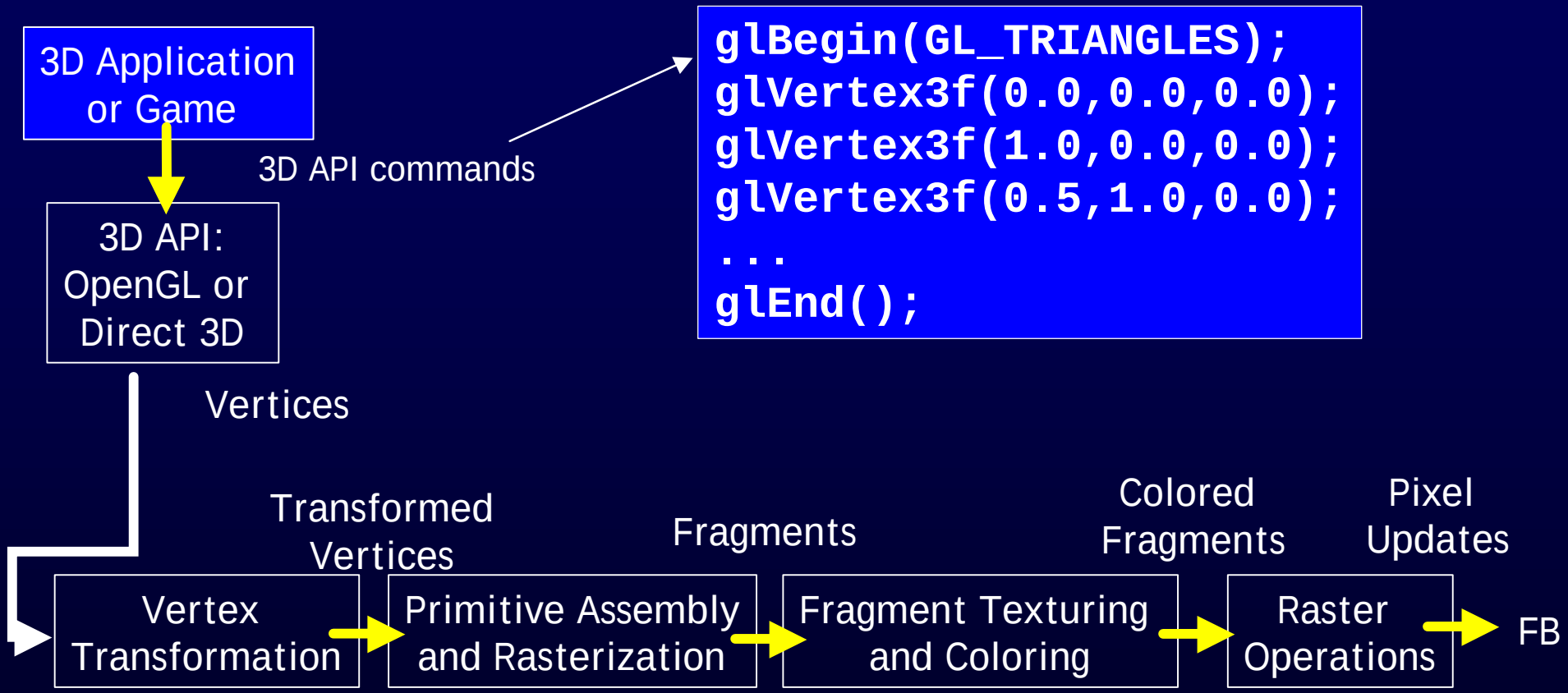
Graphics Pipeline

Joao Comba, UFRGS



Graphics Pipeline

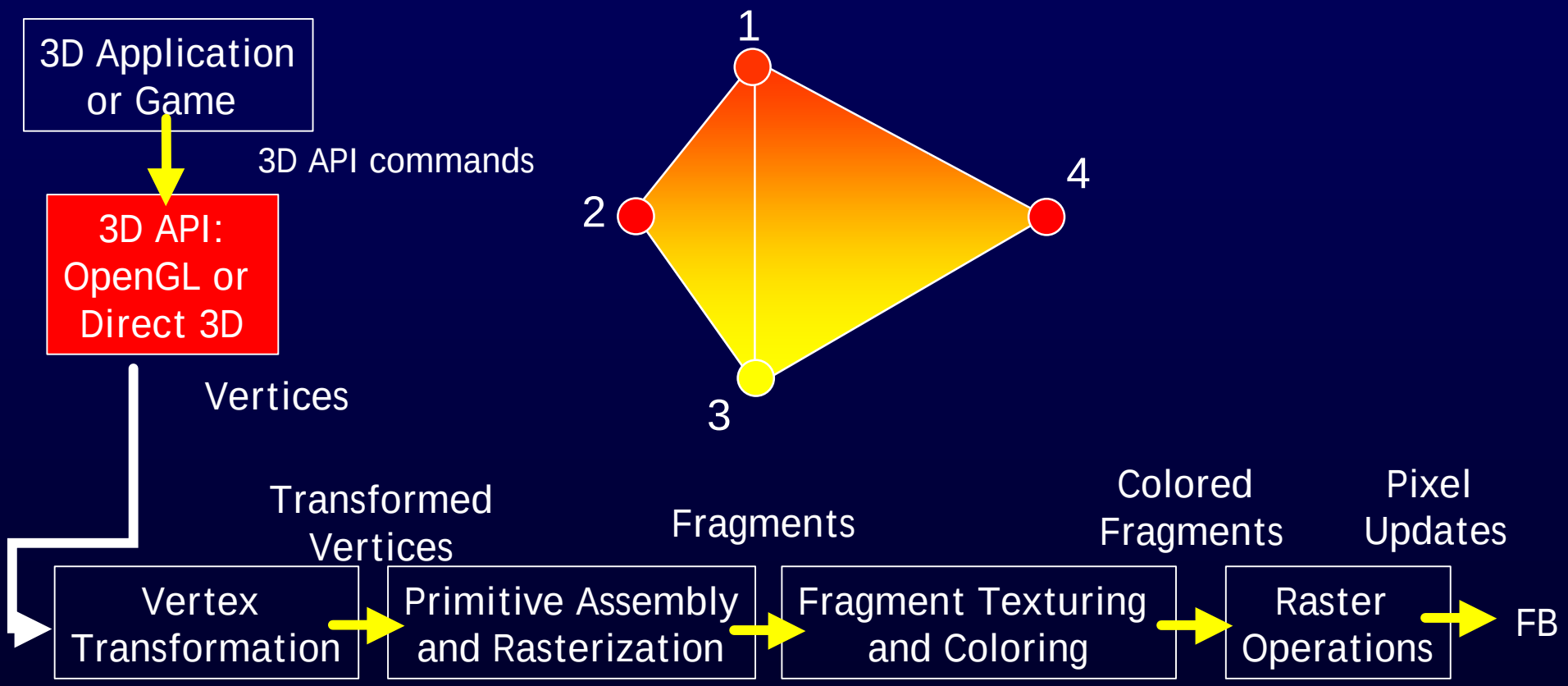
Joao Comba, UFRGS




```
glBegin(GL_TRIANGLES);
glVertex3f(0.0,0.0,0.0);
glVertex3f(1.0,0.0,0.0);
glVertex3f(0.5,1.0,0.0);
...
glEnd();
```

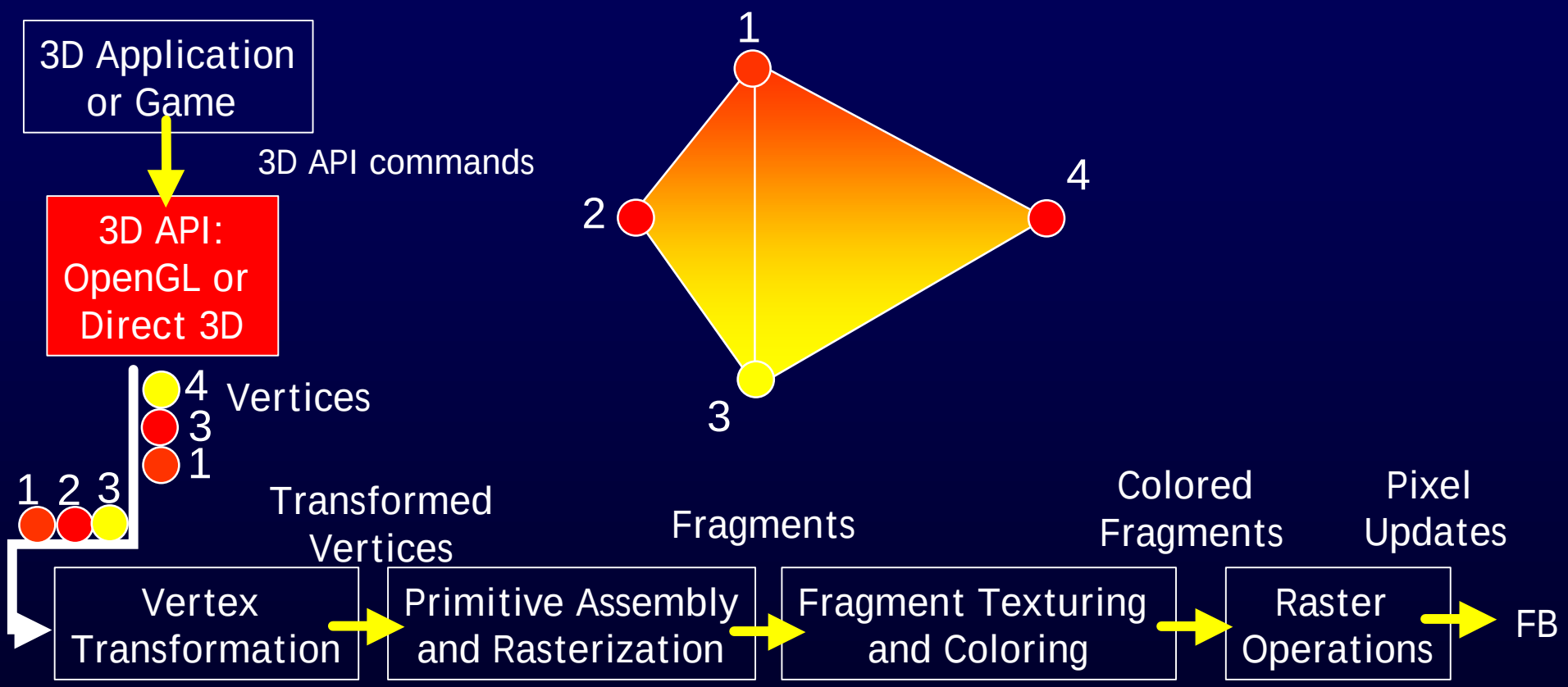
Graphics Pipeline

Joao Comba, UFRGS



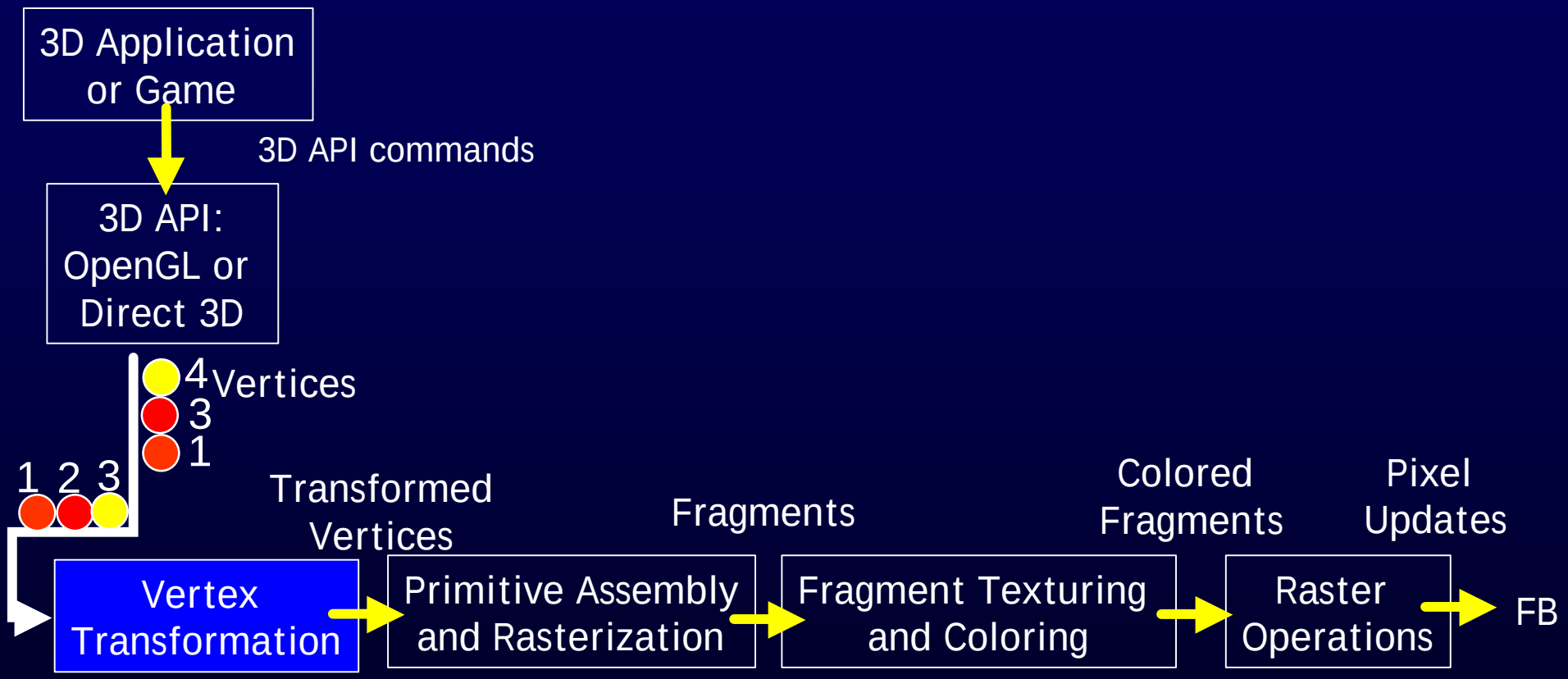
Graphics Pipeline

Joao Comba, UFRGS



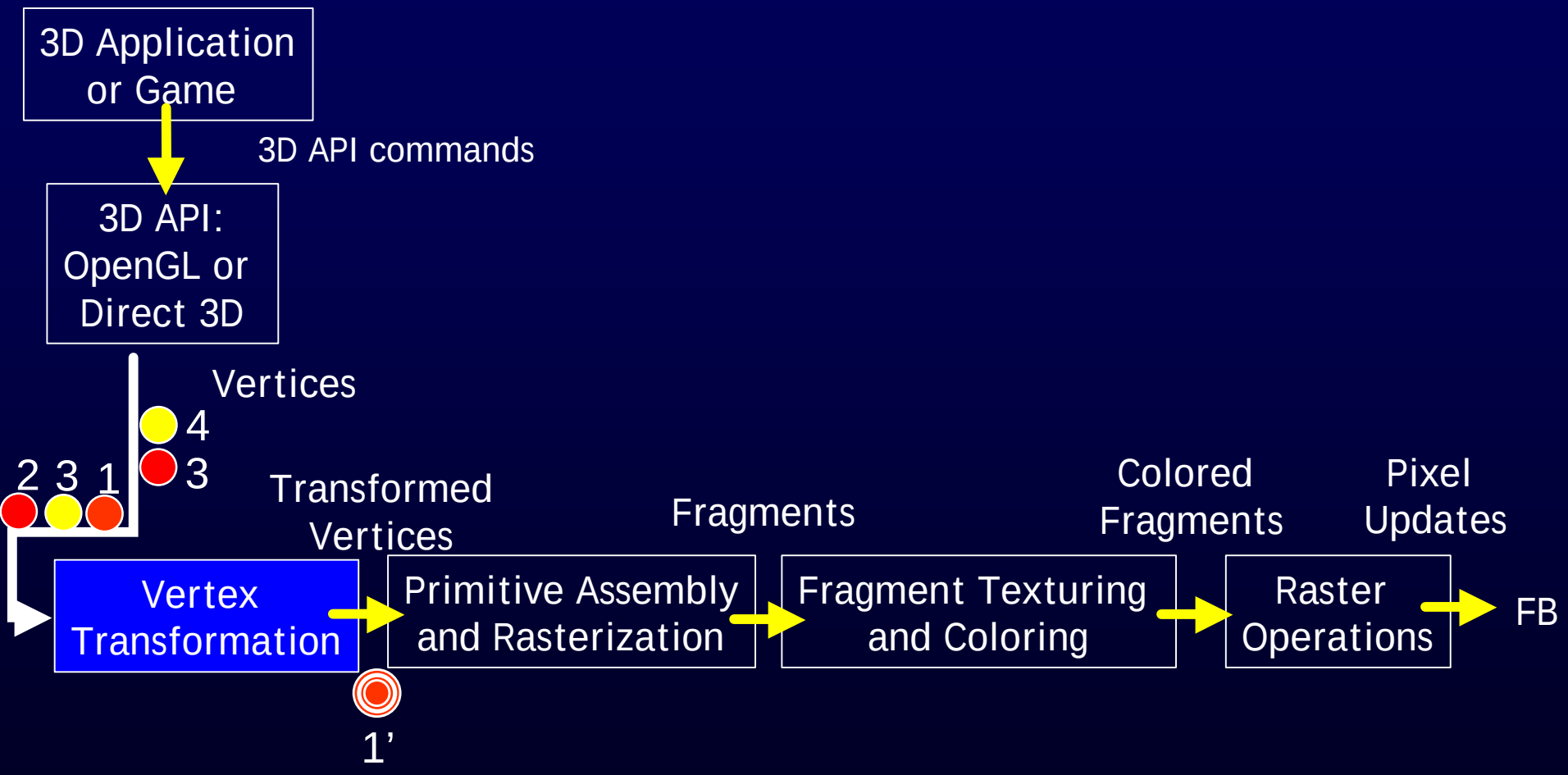
Graphics Pipeline

Joao Comba, UFRGS



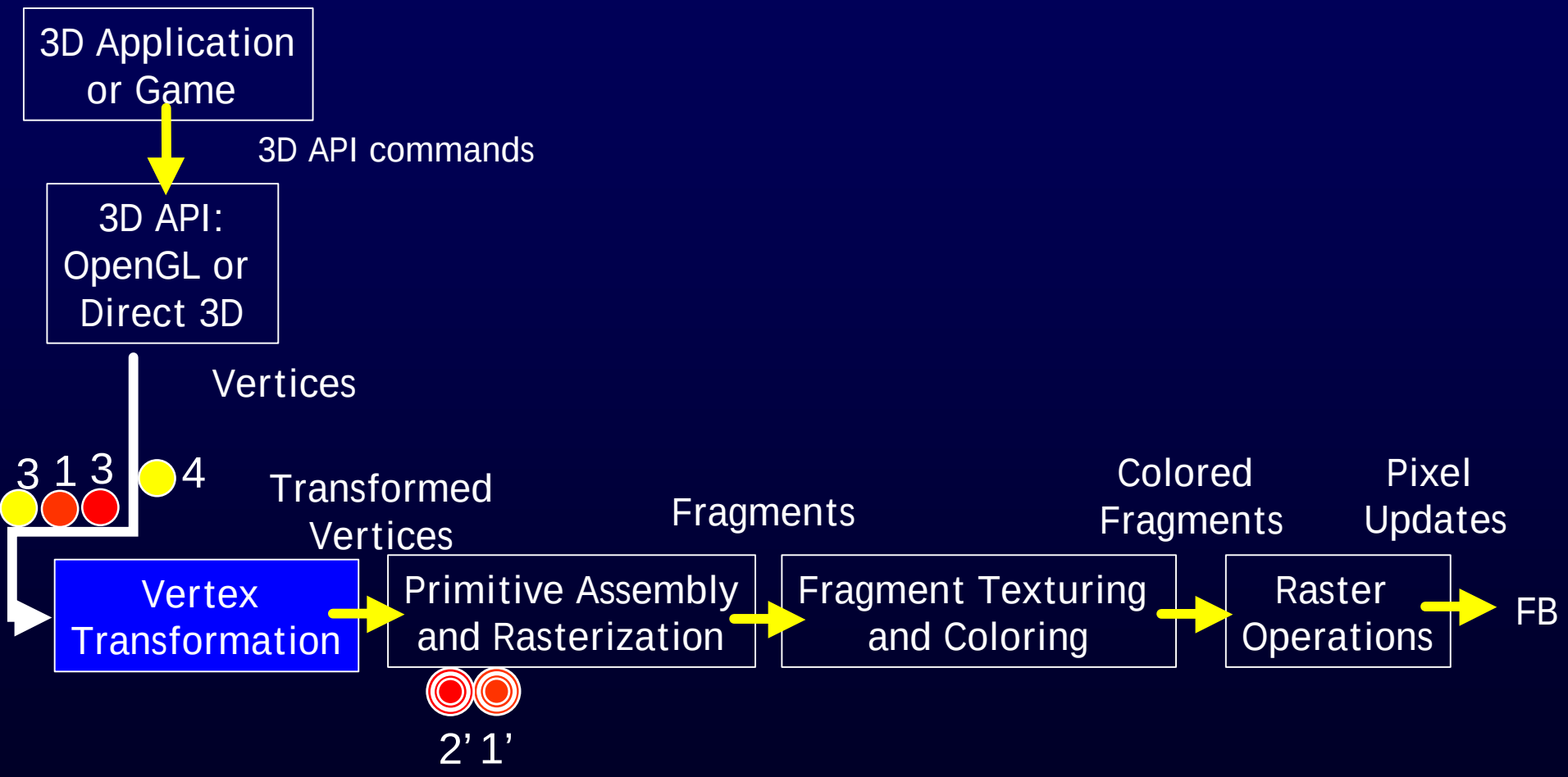
Graphics Pipeline

Joao Comba, UFRGS



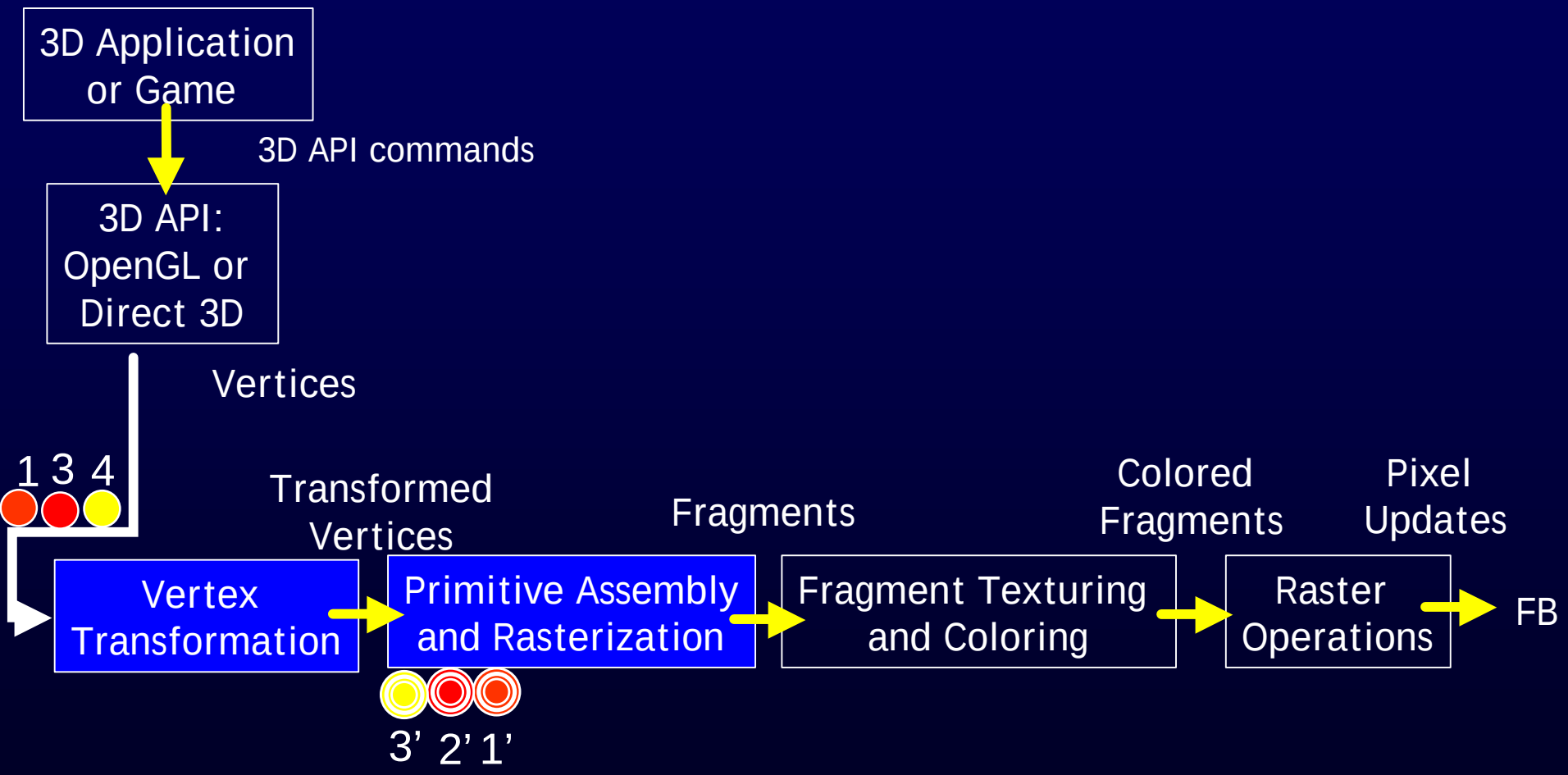
Graphics Pipeline

Joao Comba, UFRGS



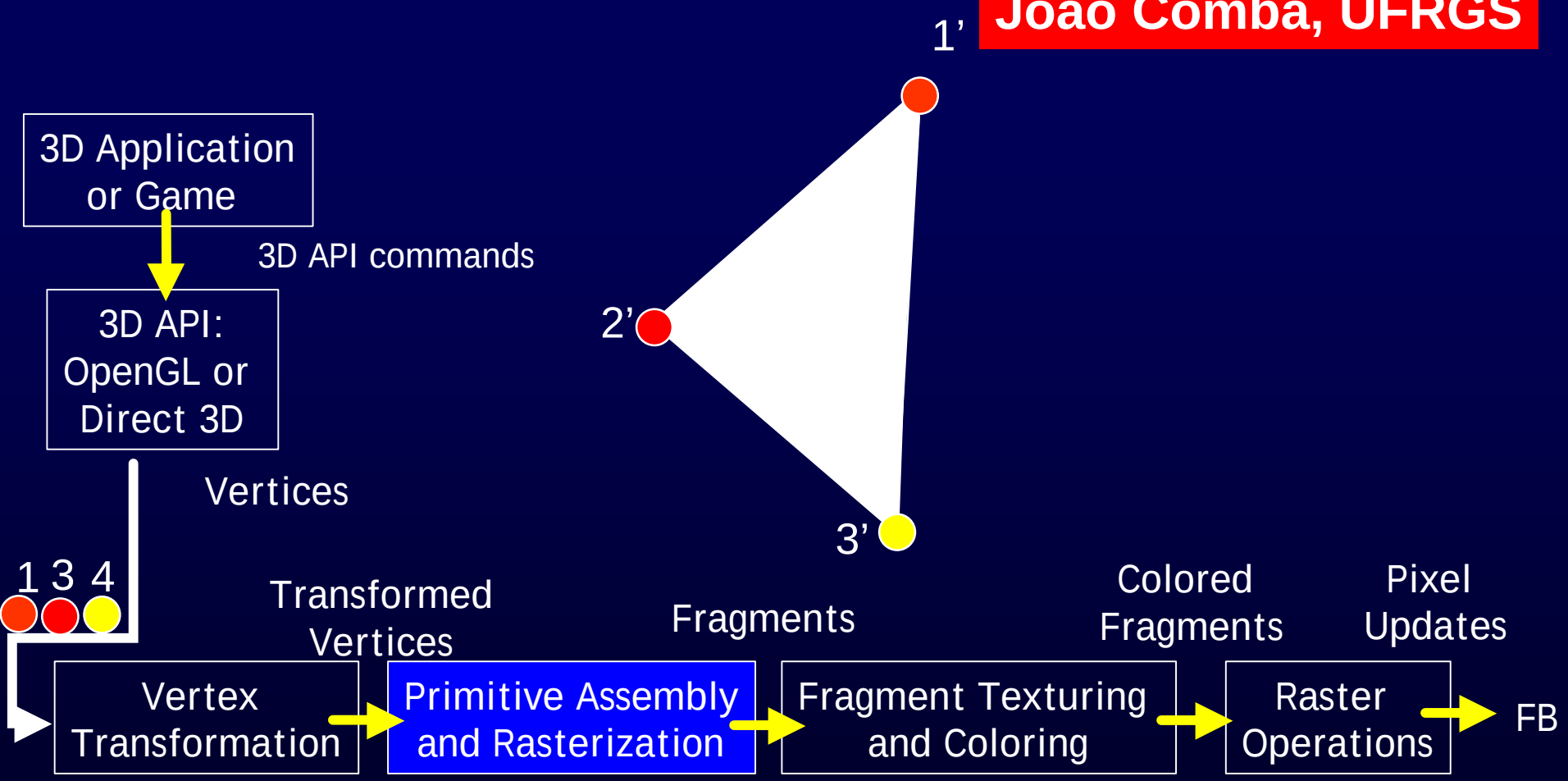
Graphics Pipeline

Joao Comba, UFRGS



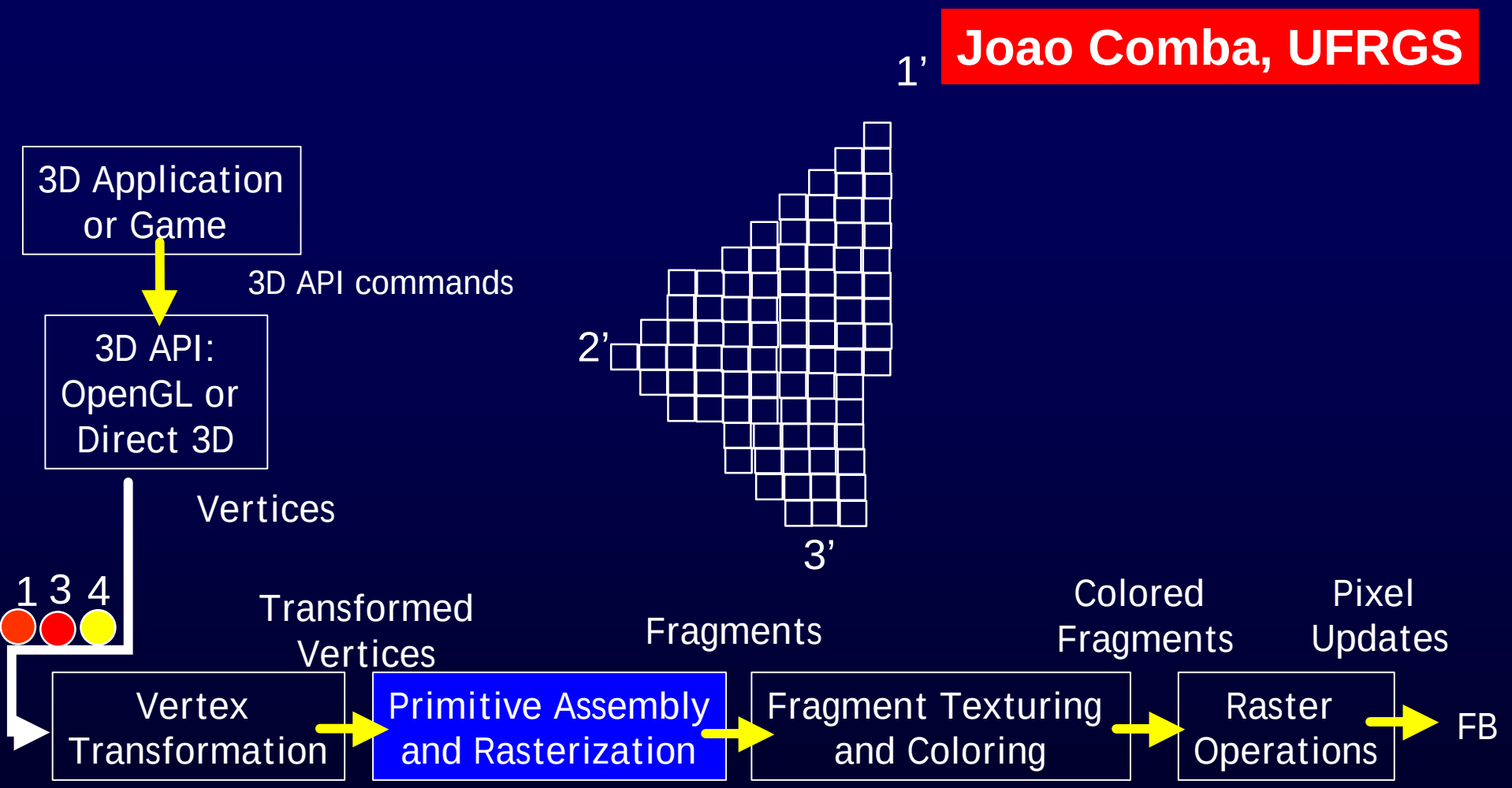
Graphics Pipeline

Joao Comba, UFRGS



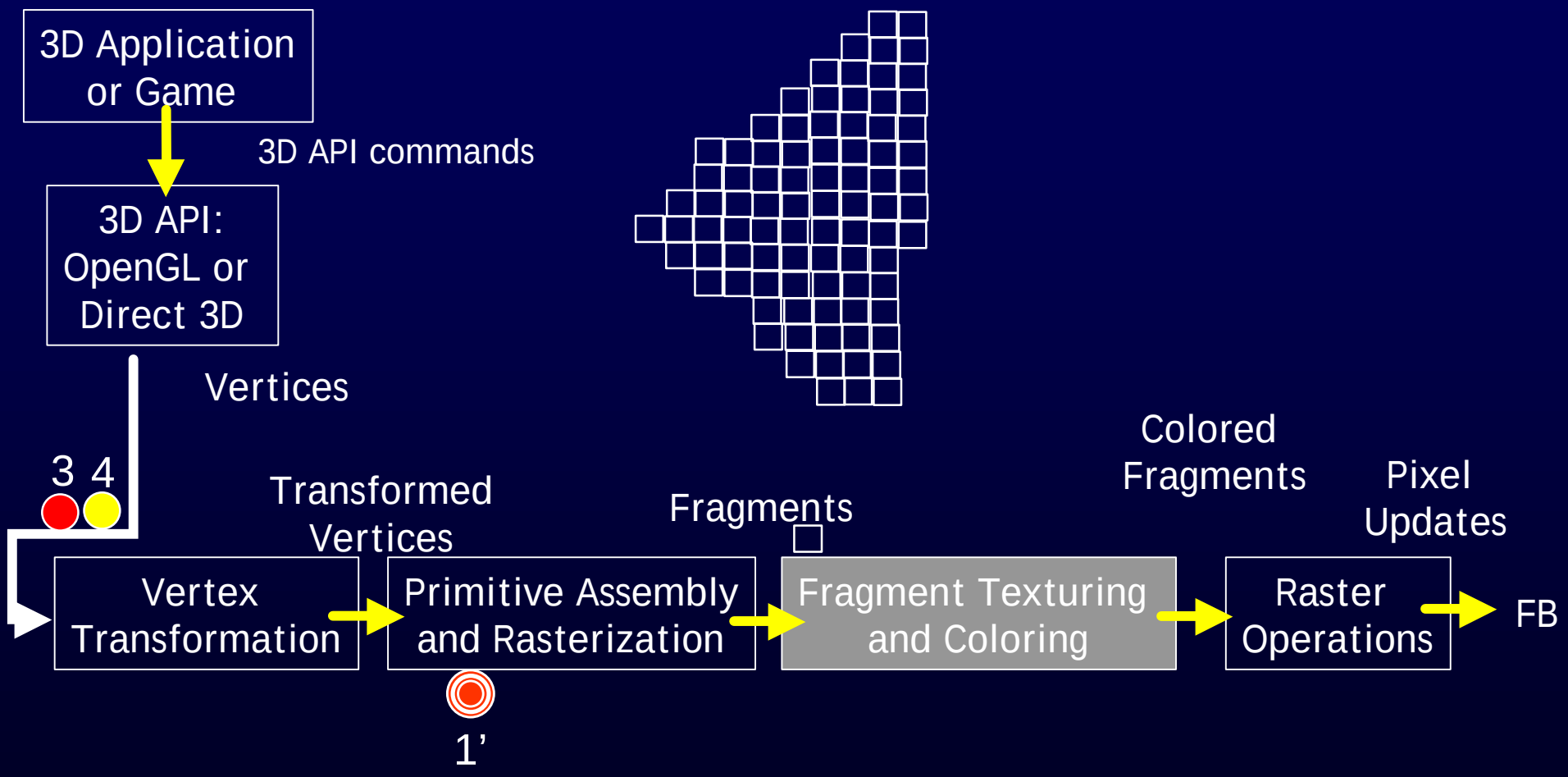
Graphics Pipeline

Joao Comba, UFRGS



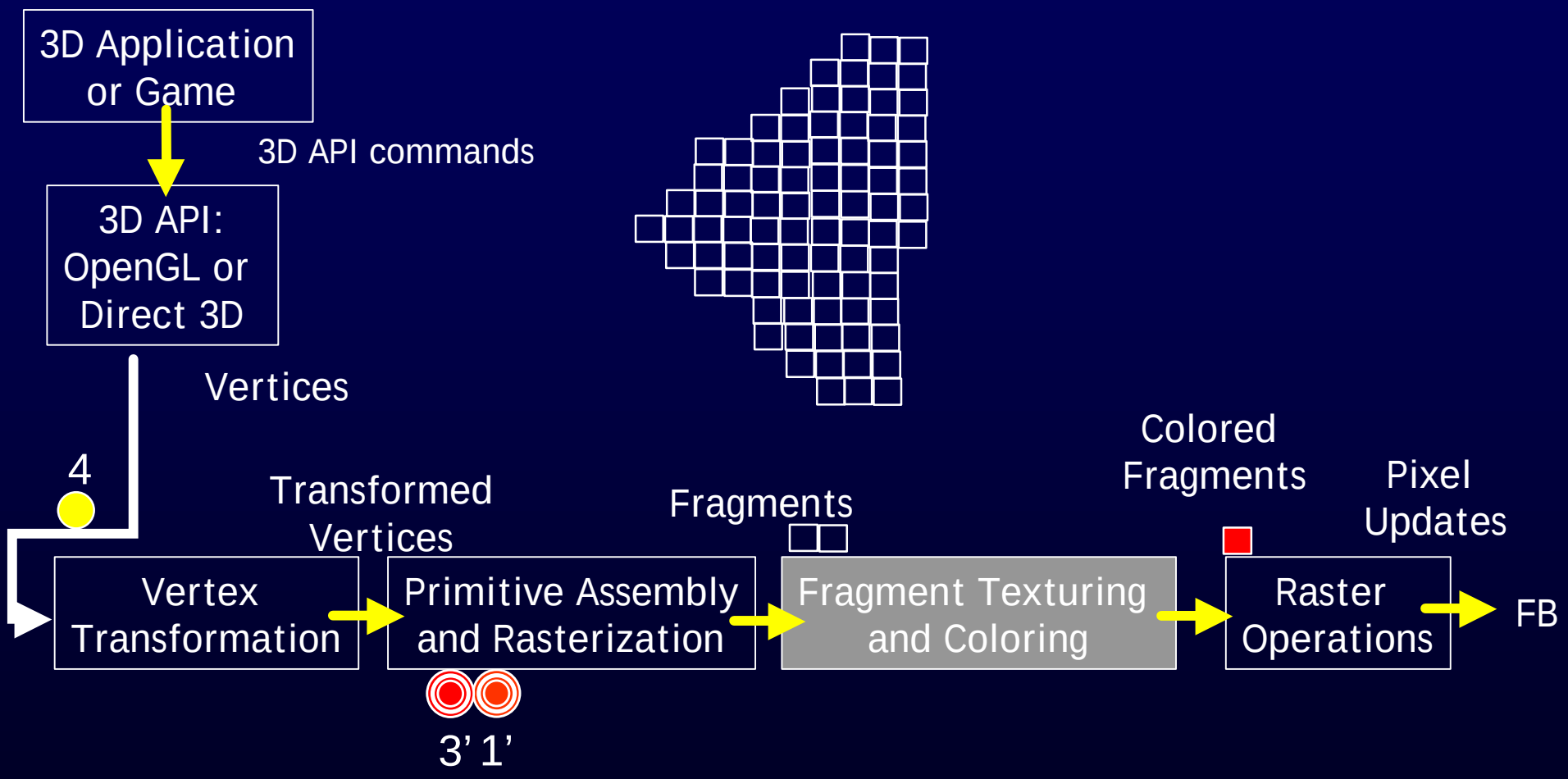
Graphics Pipeline

Joao Comba, UFRGS



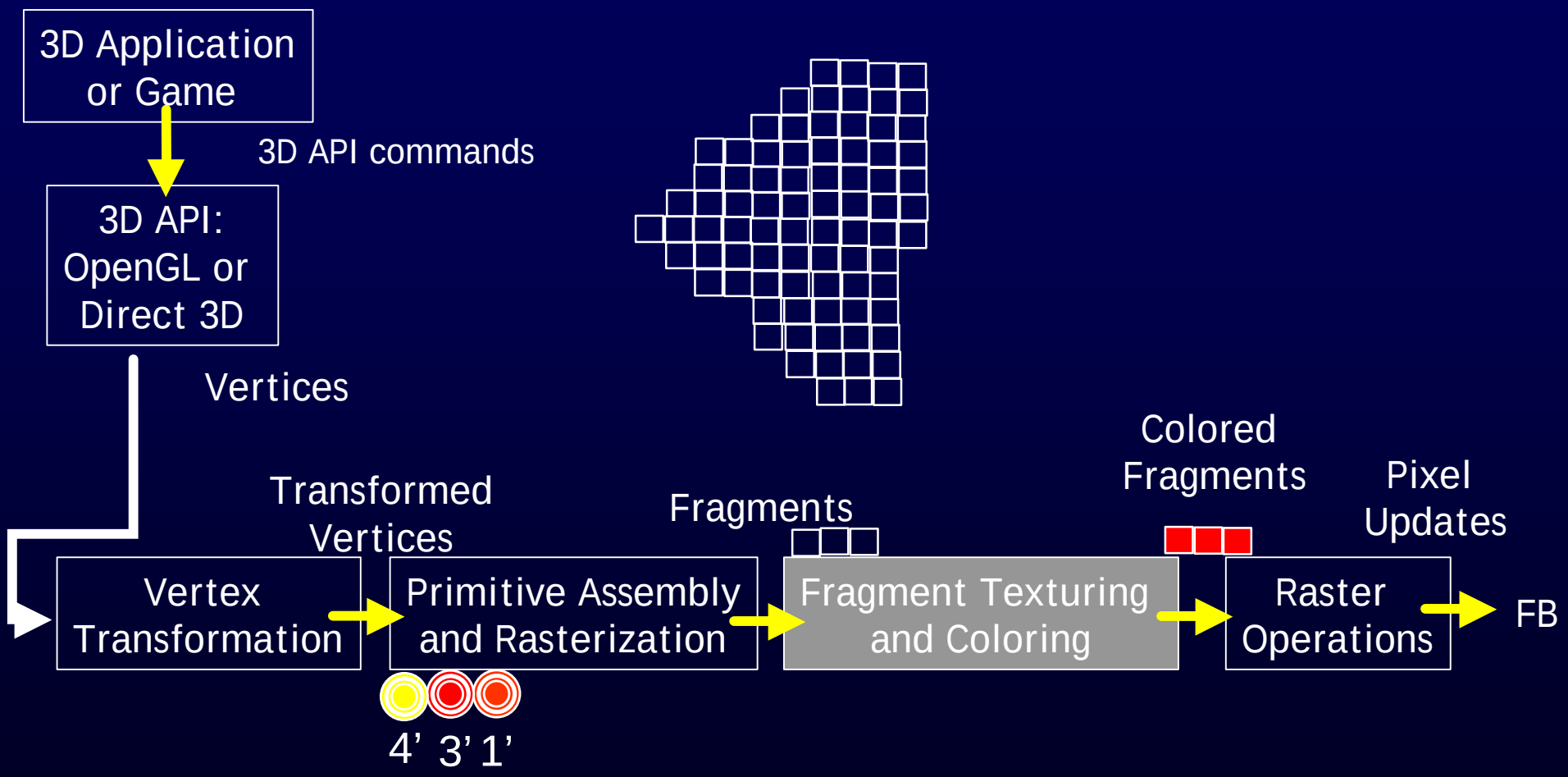
Graphics Pipeline

Joao Comba, UFRGS



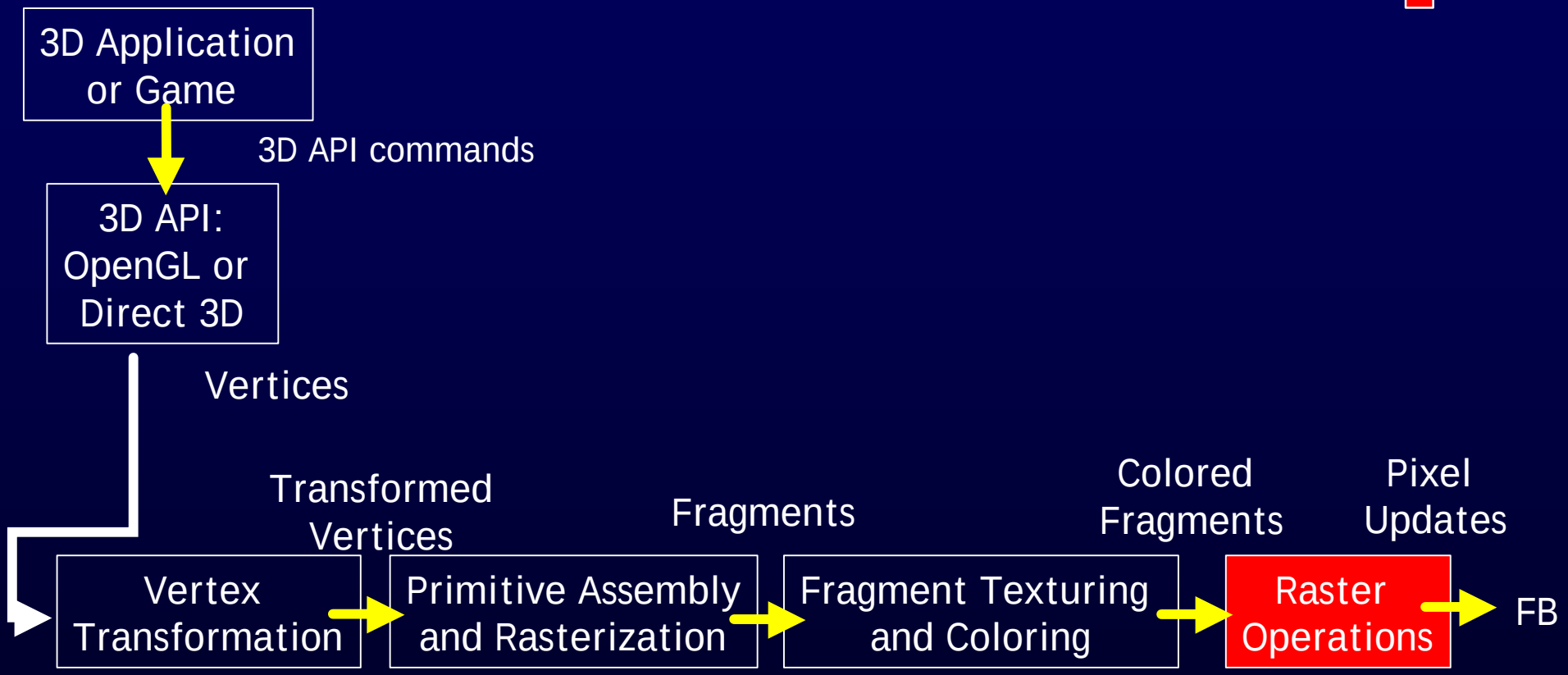
Graphics Pipeline

Joao Comba, UFRGS



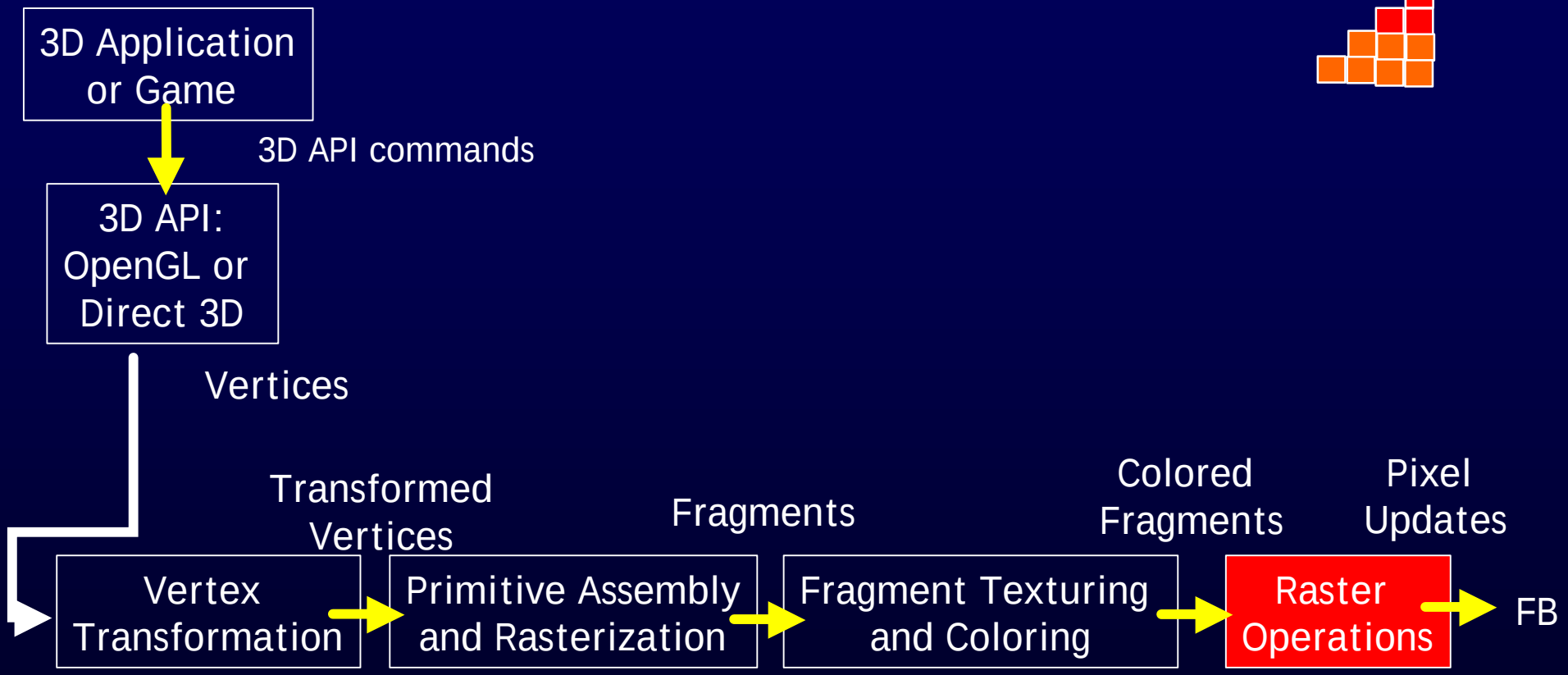
Graphics Pipeline

Joao Comba, UFRGS



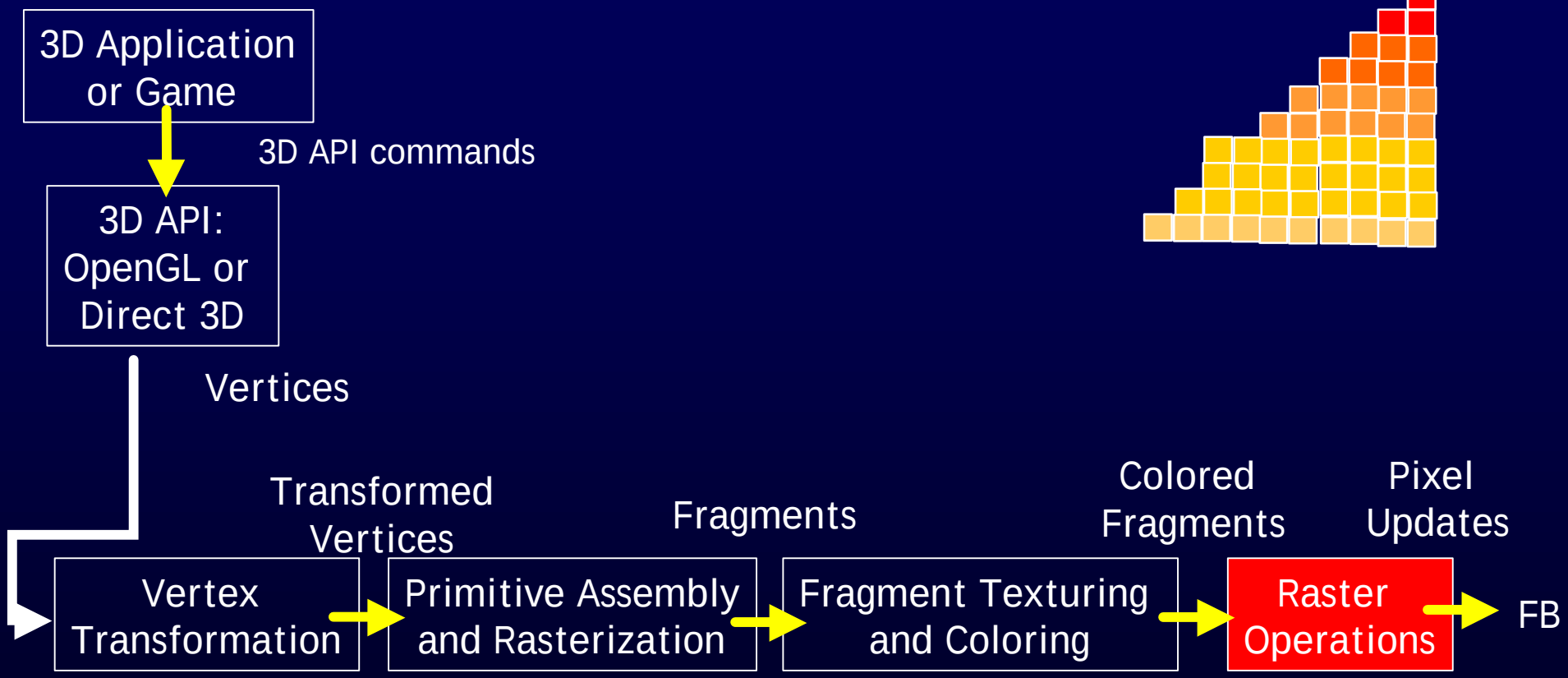
Graphics Pipeline

Joao Comba, UFRGS



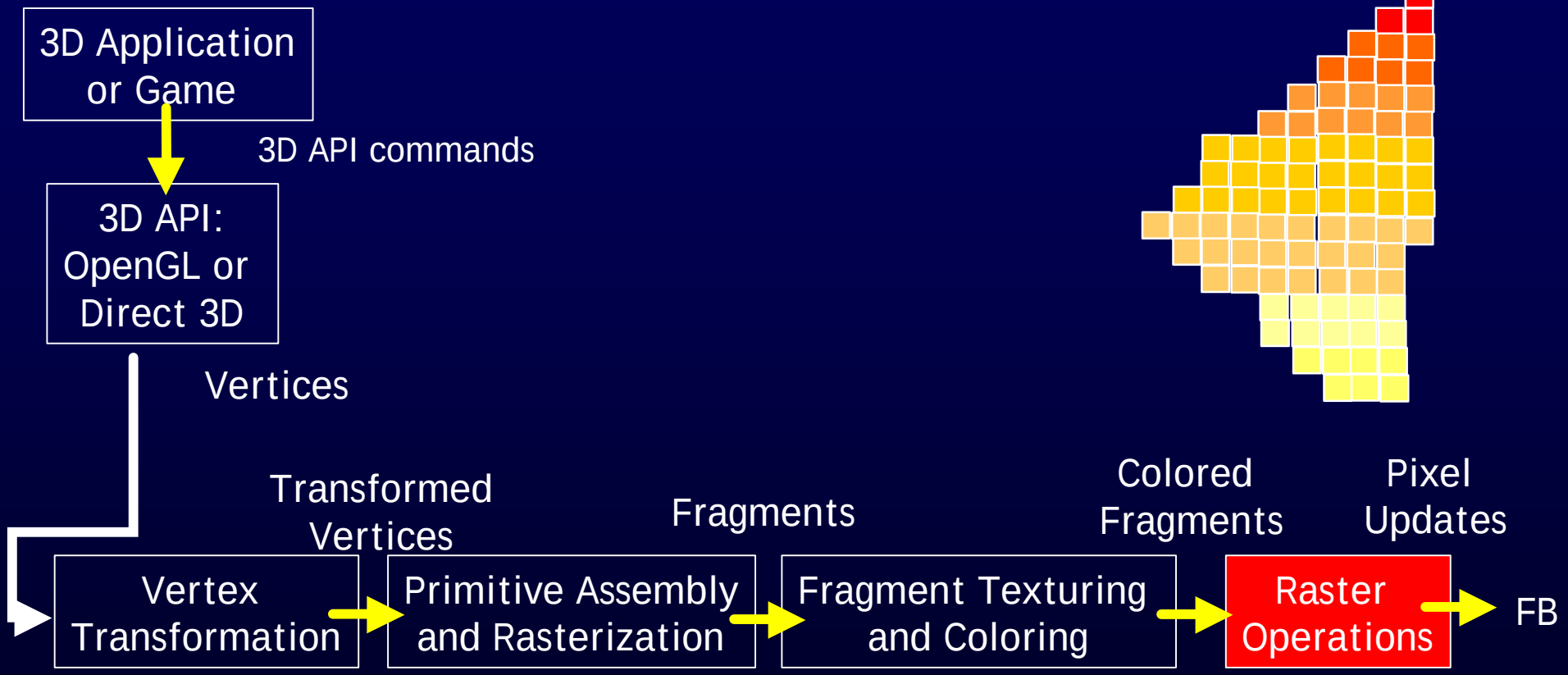
Graphics Pipeline

Joao Comba, UFRGS



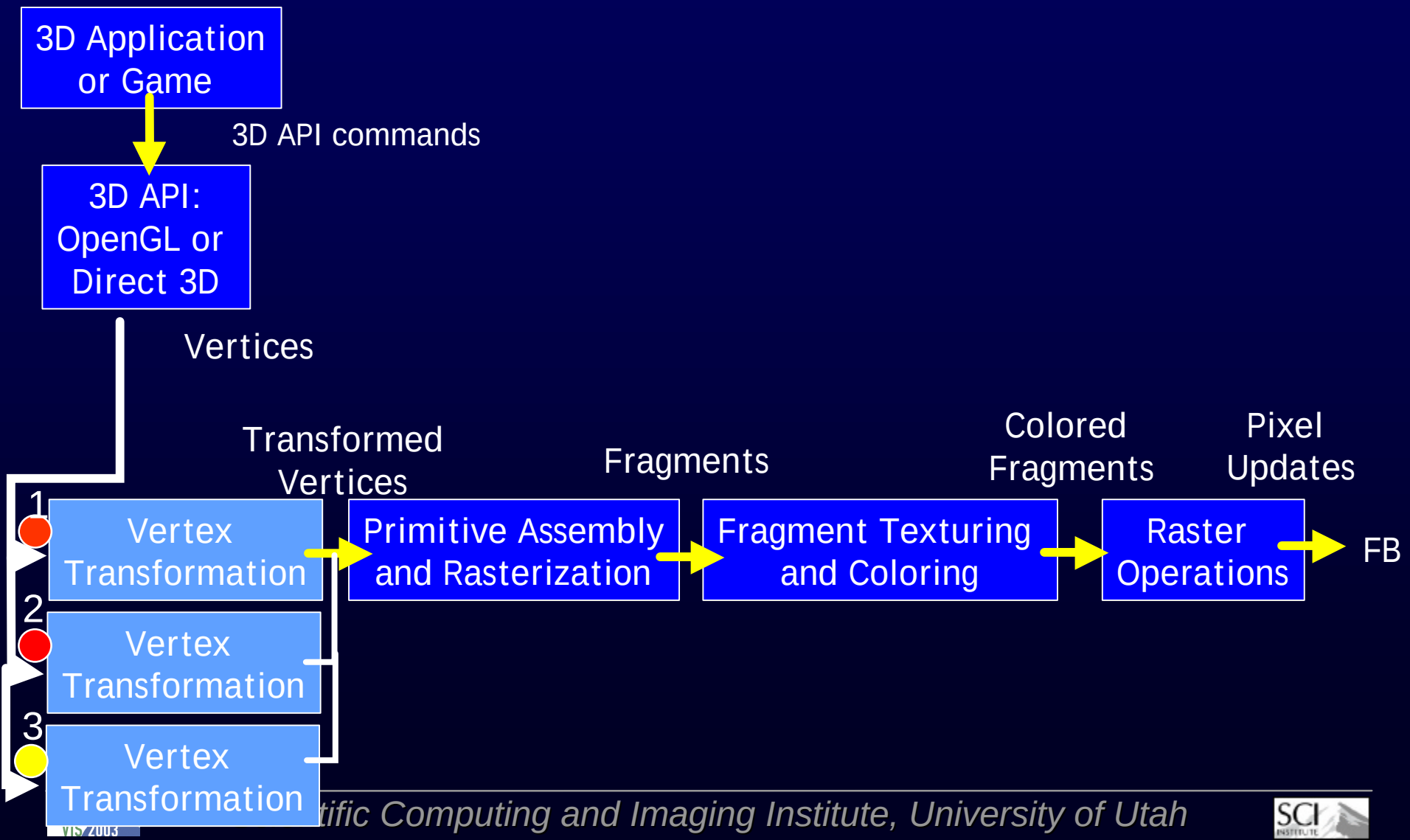
Graphics Pipeline

Joao Comba, UFRGS



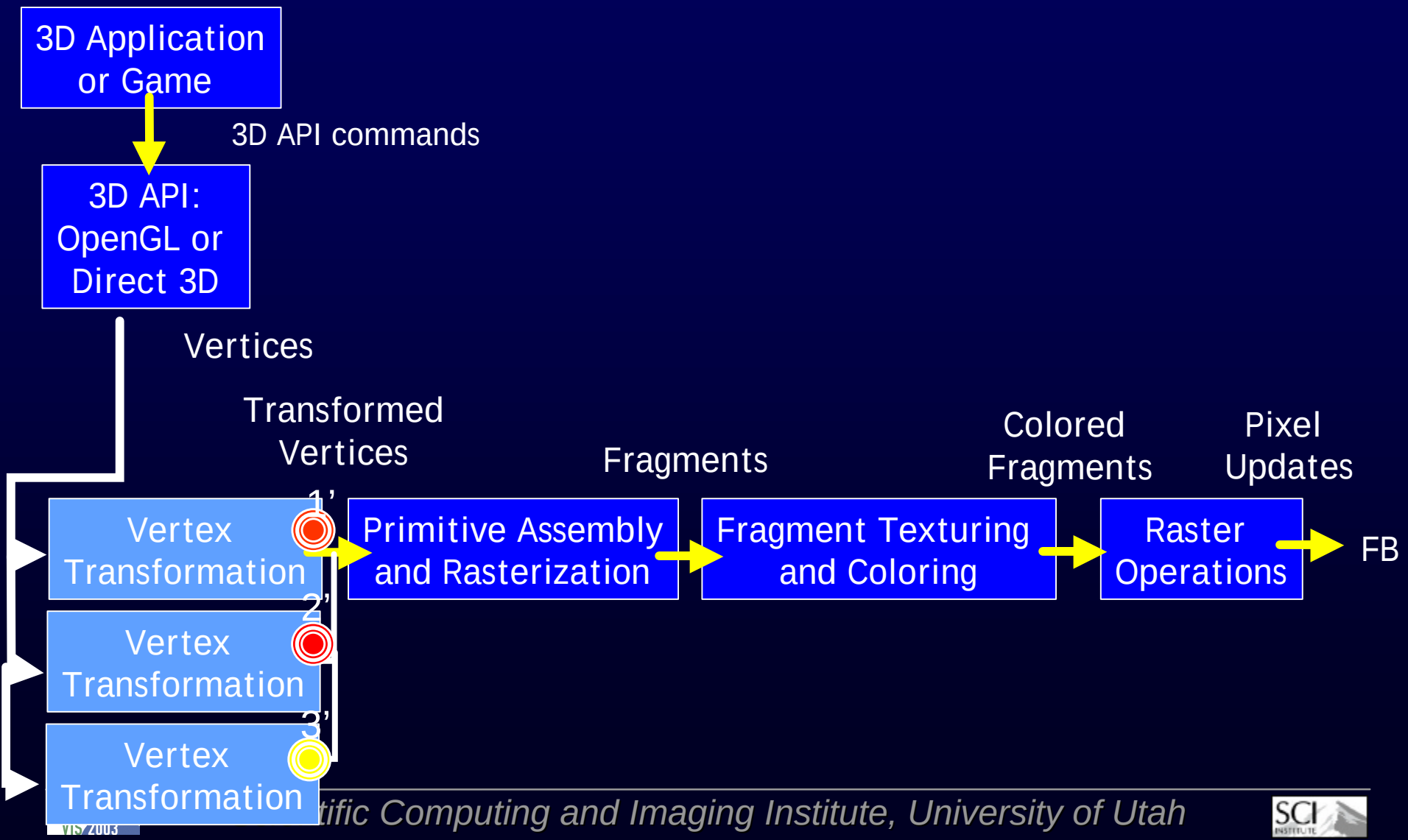
Graphics Pipeline: Parallelization

Joao Comba, UFRGS



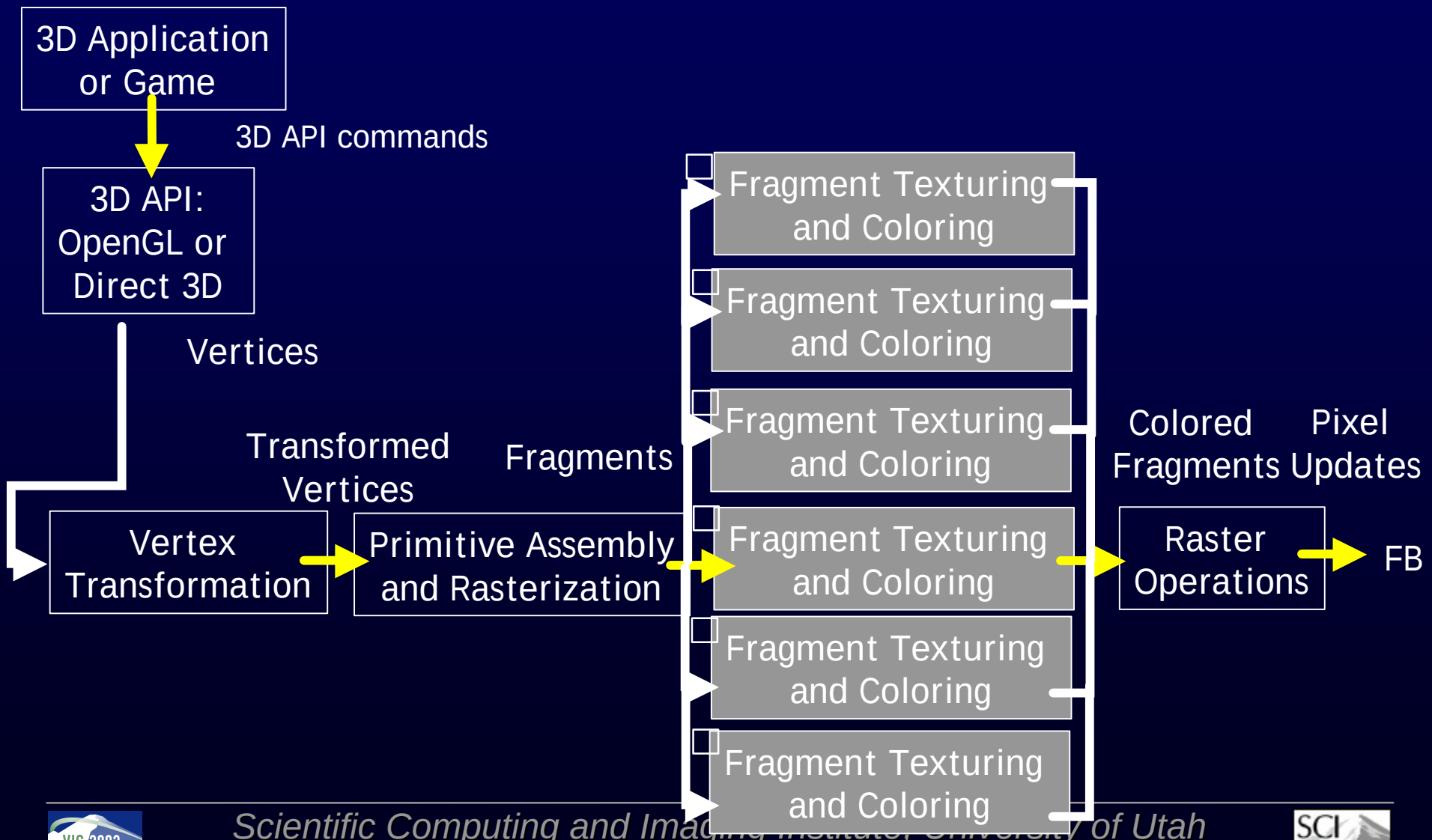
Graphics Pipeline: Parallelization

Joao Comba, UFRGS



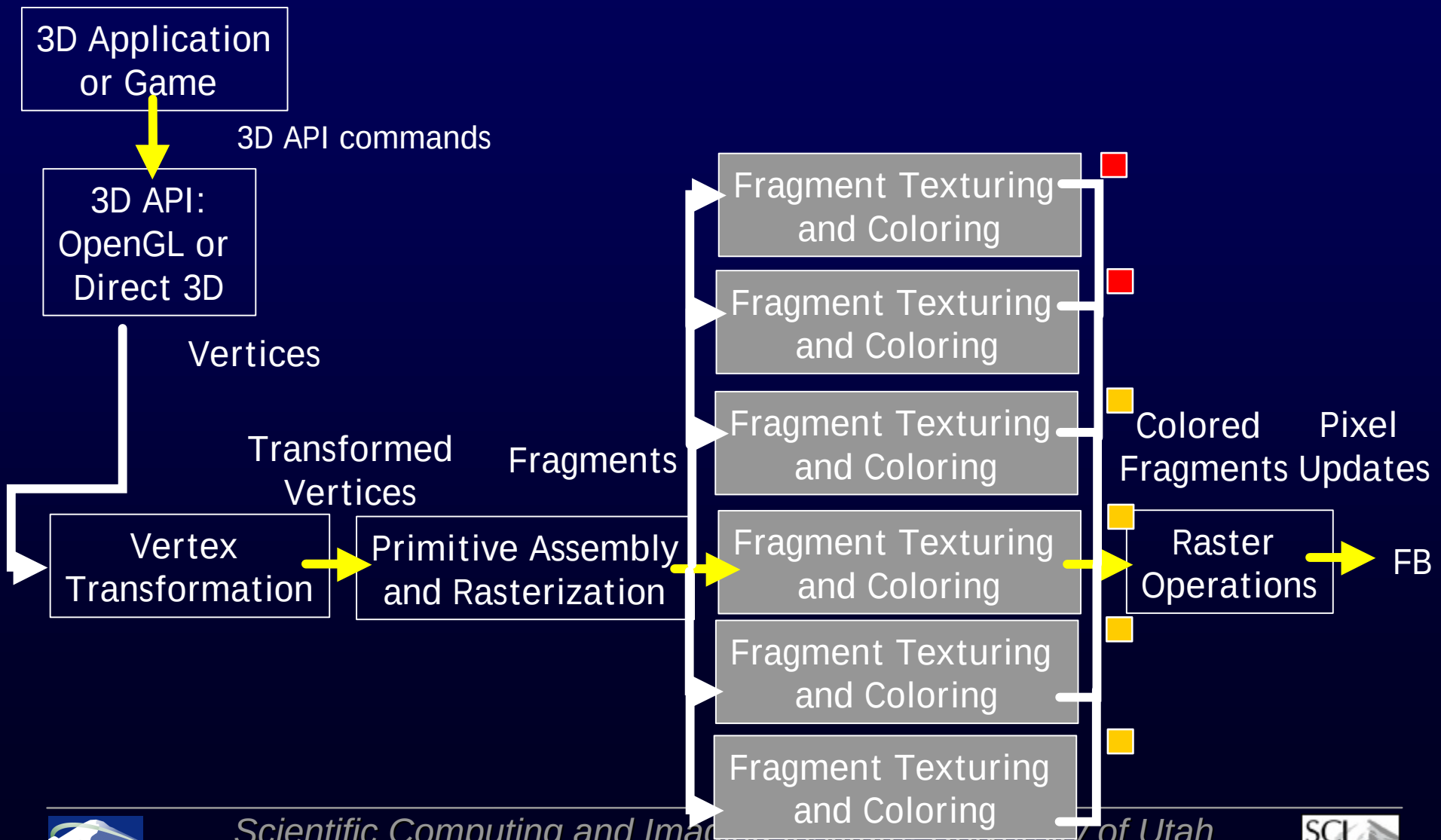
Graphics Pipeline: Parallelization

Joao Comba, UFRGS



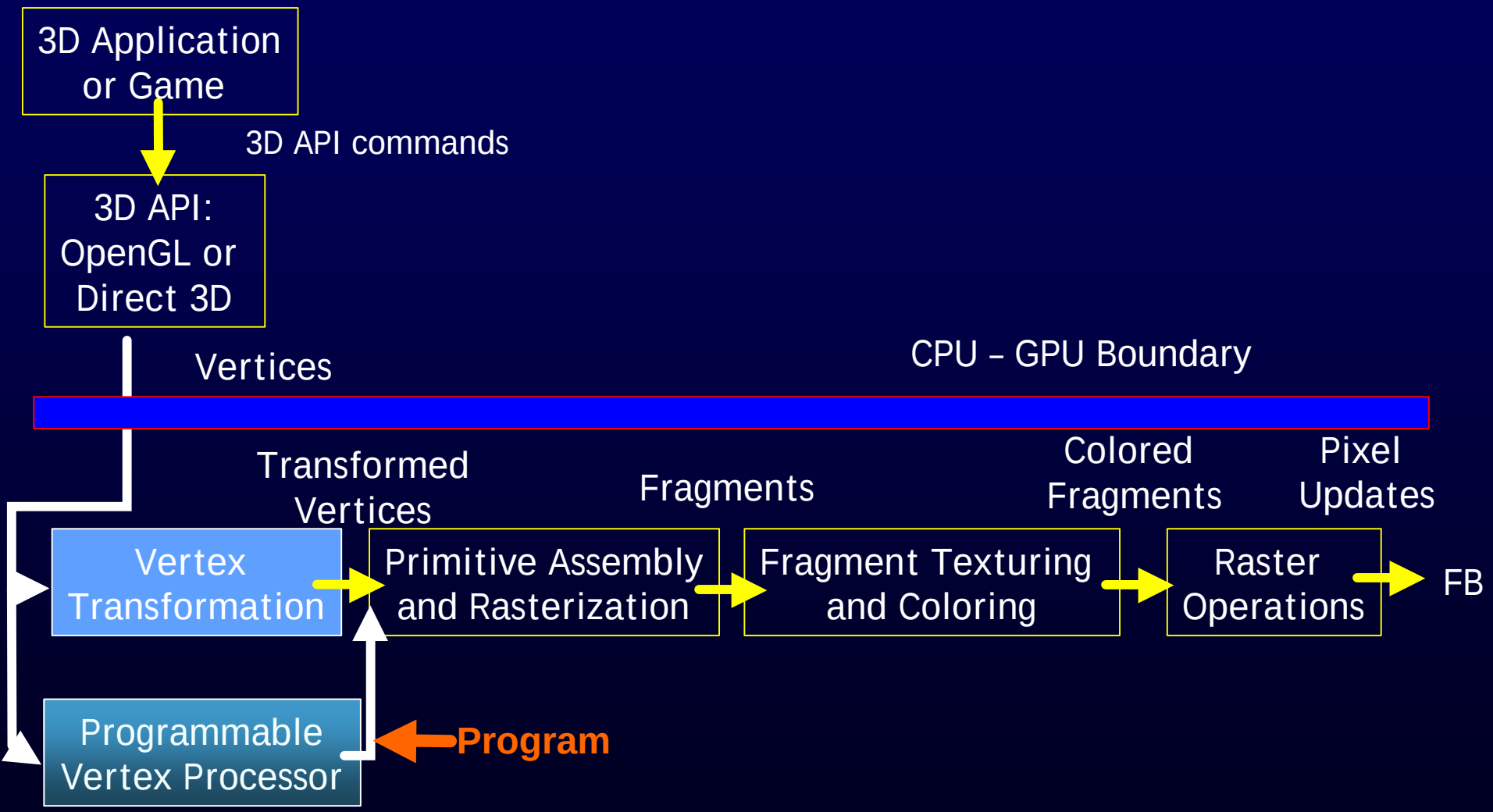
Graphics Pipeline: Parallelization

Joao Comba, UFRGS



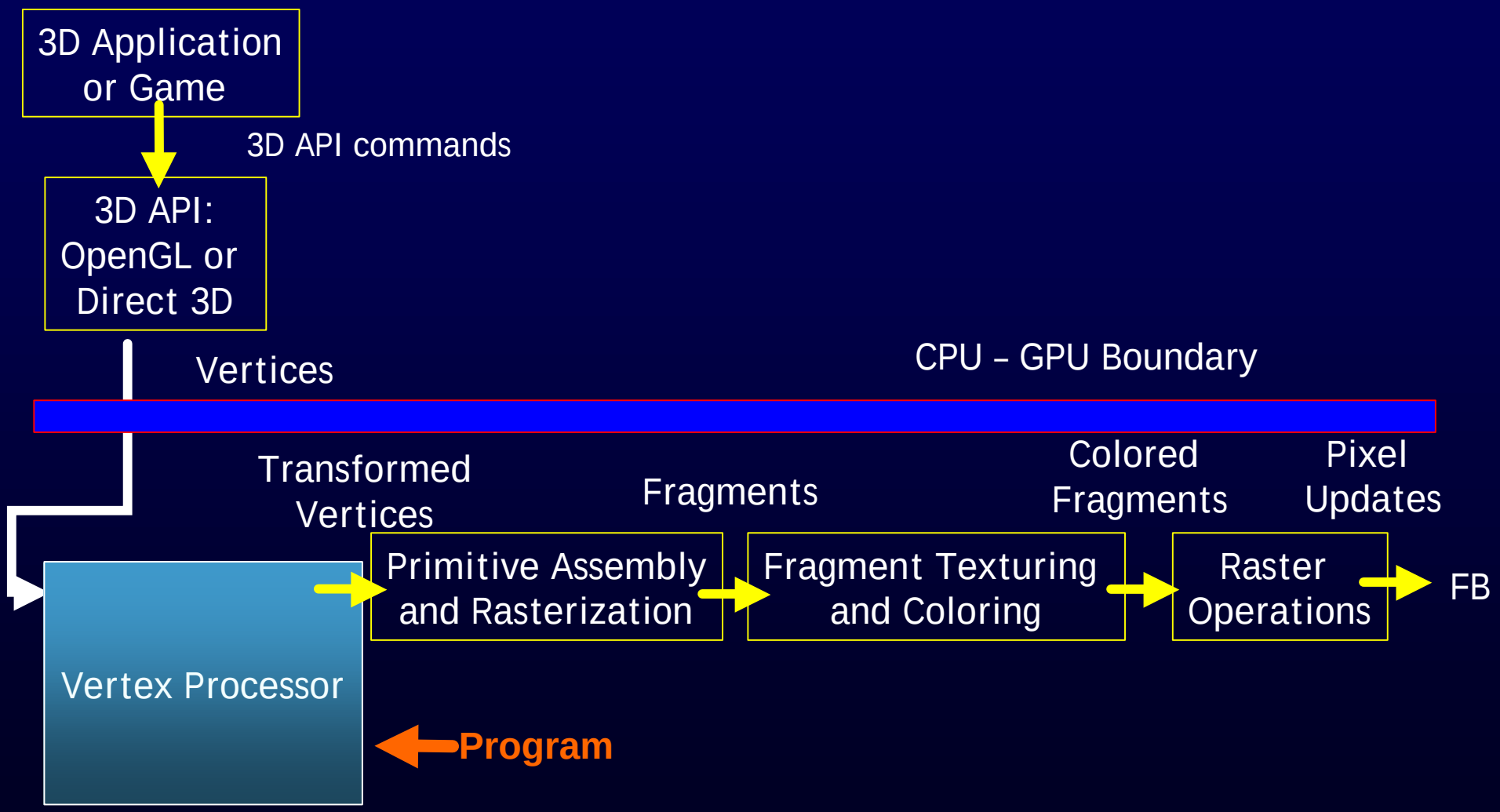
Programmable Graphics Pipeline

Joao Comba, UFRGS



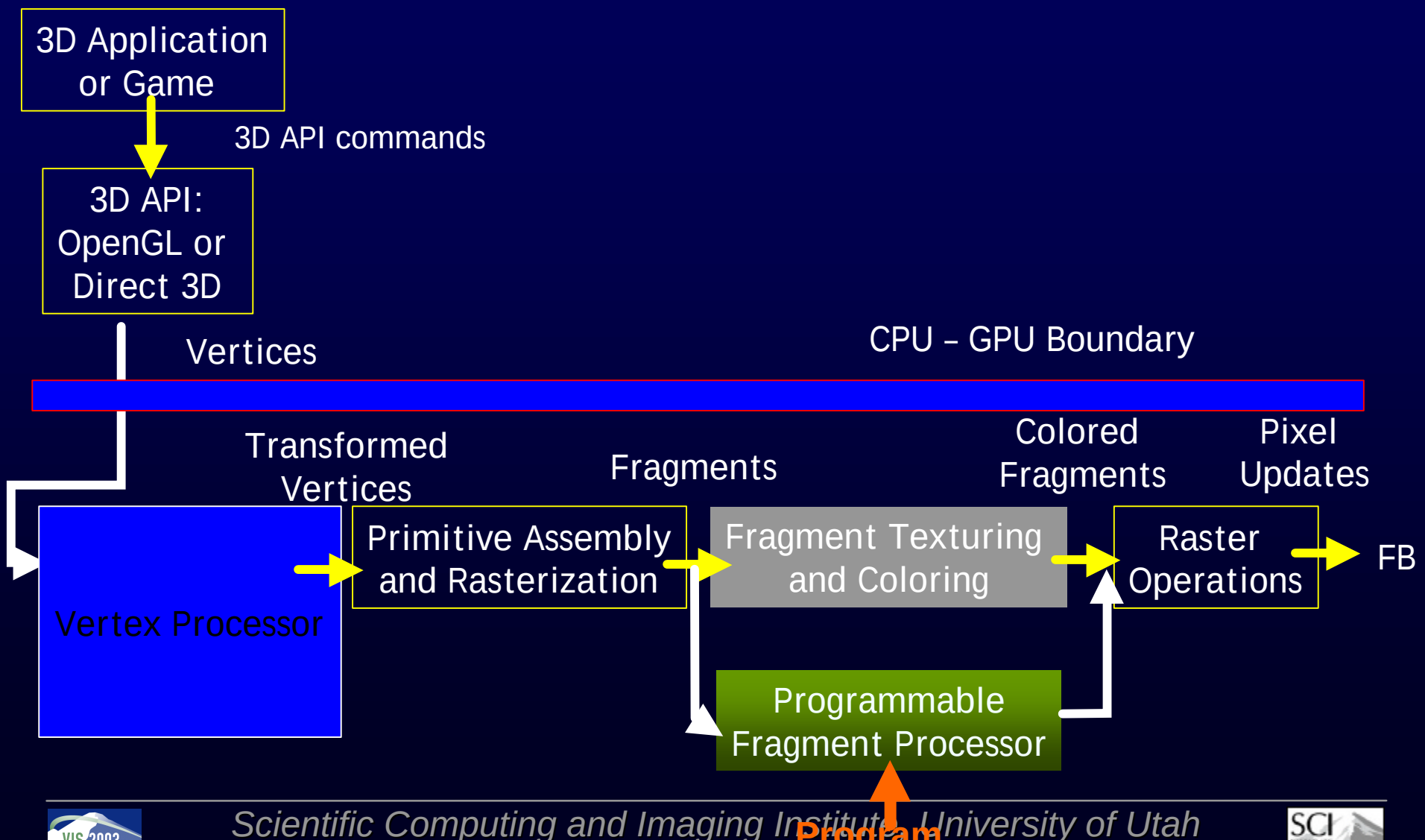
Programmable Graphics Pipeline

Joao Comba, UFRGS



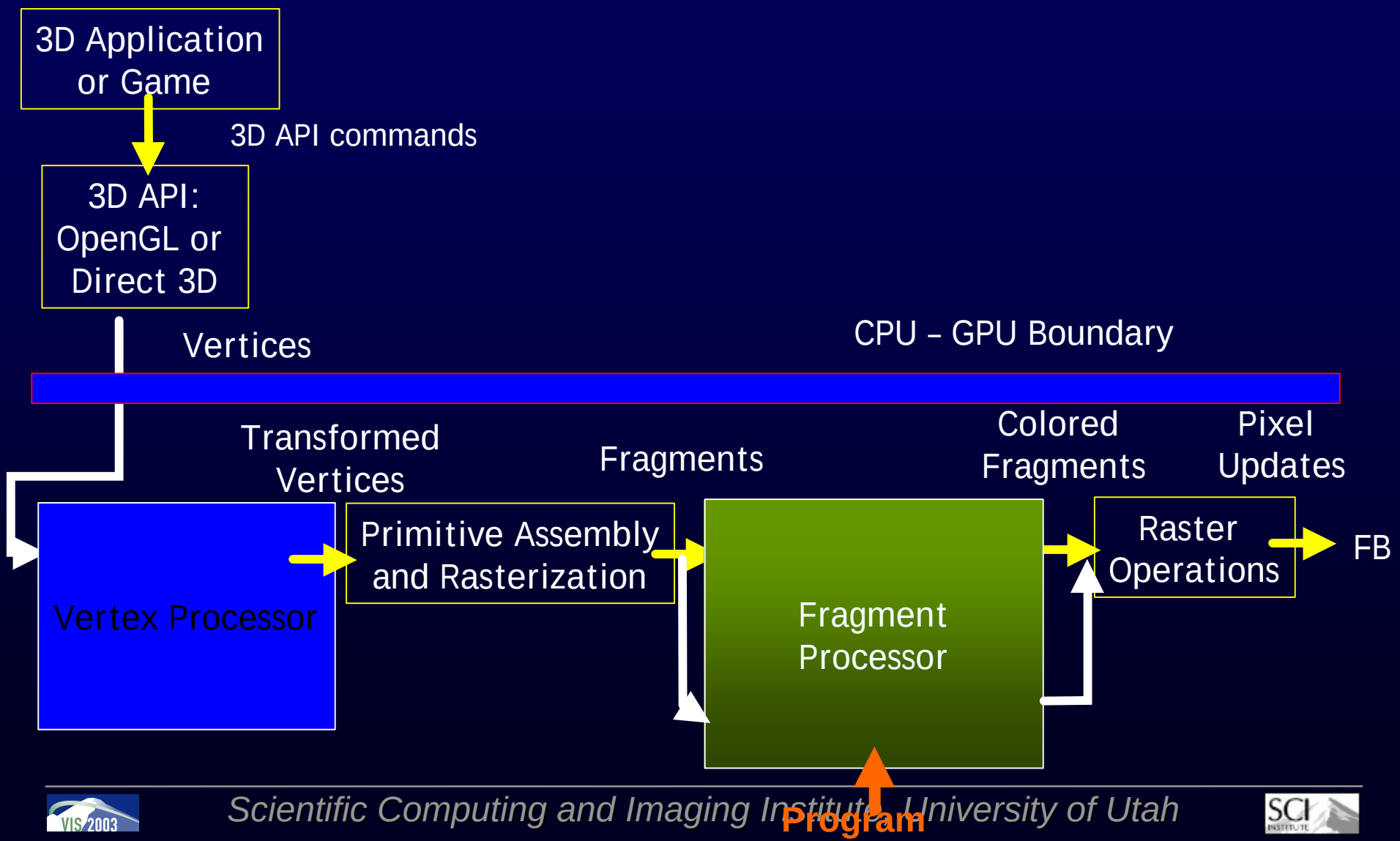
Programmable Graphics Pipeline

Joao Comba, UFRGS



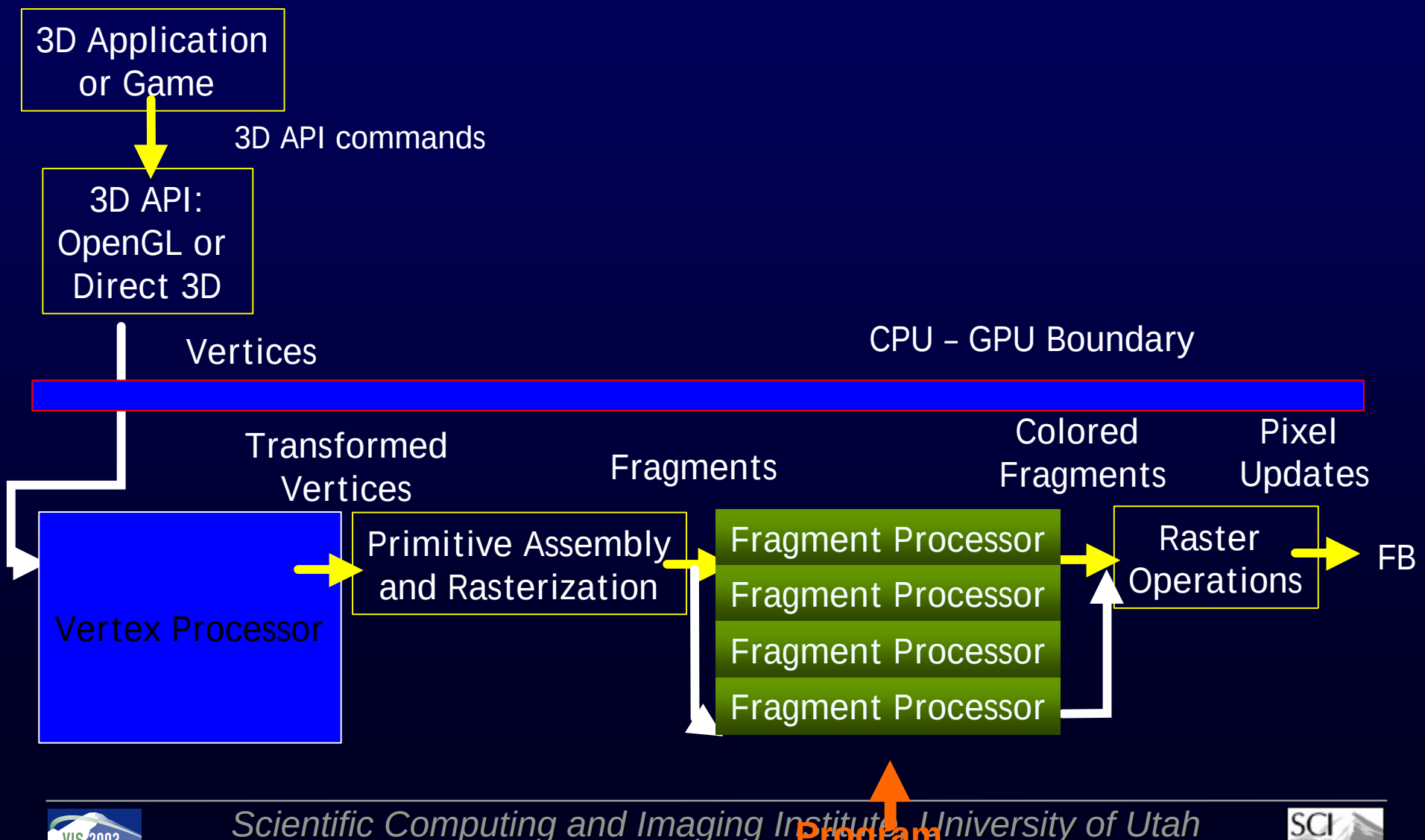
Programmable Graphics Pipeline

Joao Comba, UFRGS



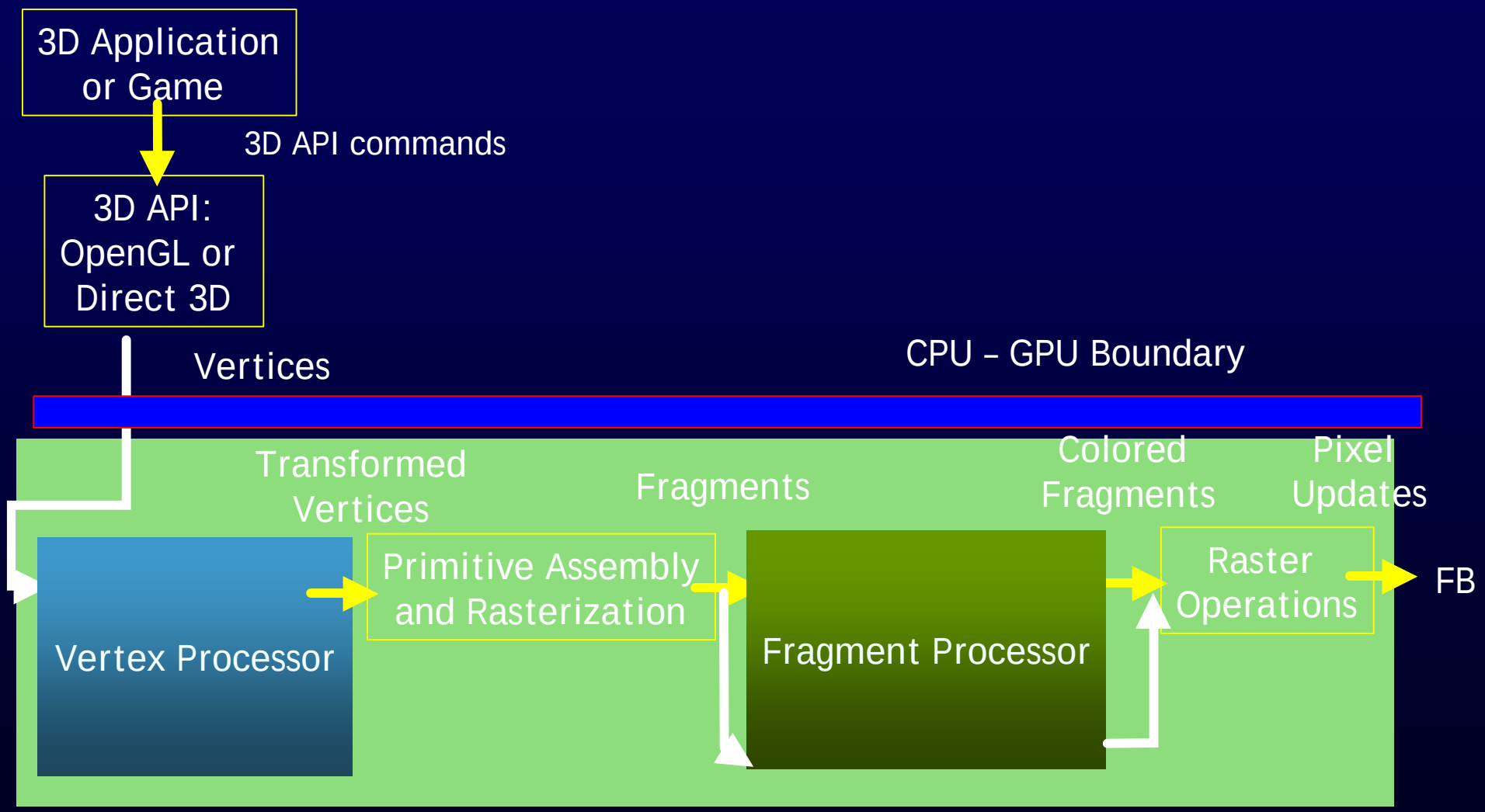
Programmable Graphics Pipeline

Joao Comba, UFRGS



Graphics Pipeline: The Big Picture

Joao Comba, UFRGS

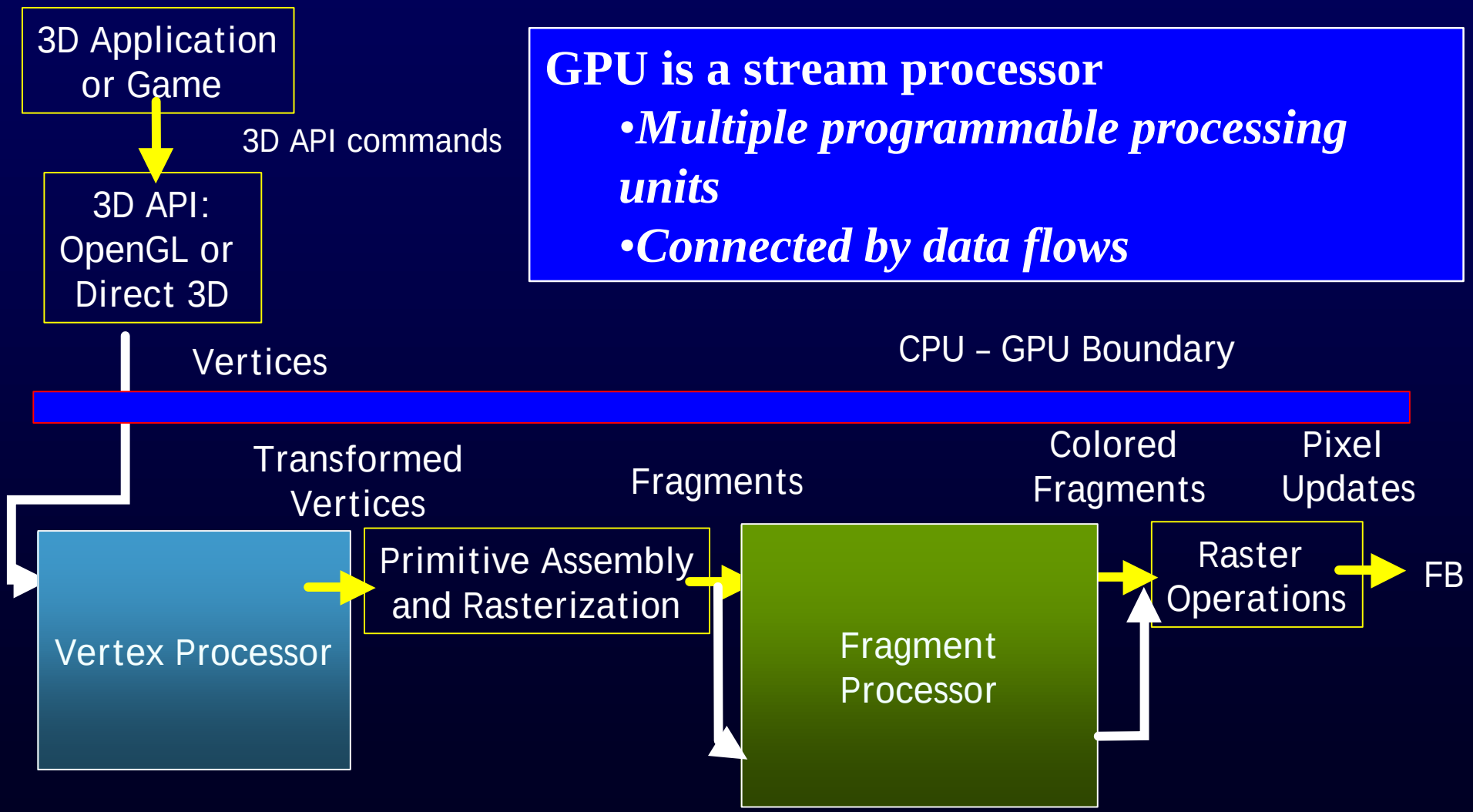


Graphics Pipeline: The Big Picture

Joao Comba, UFRGS

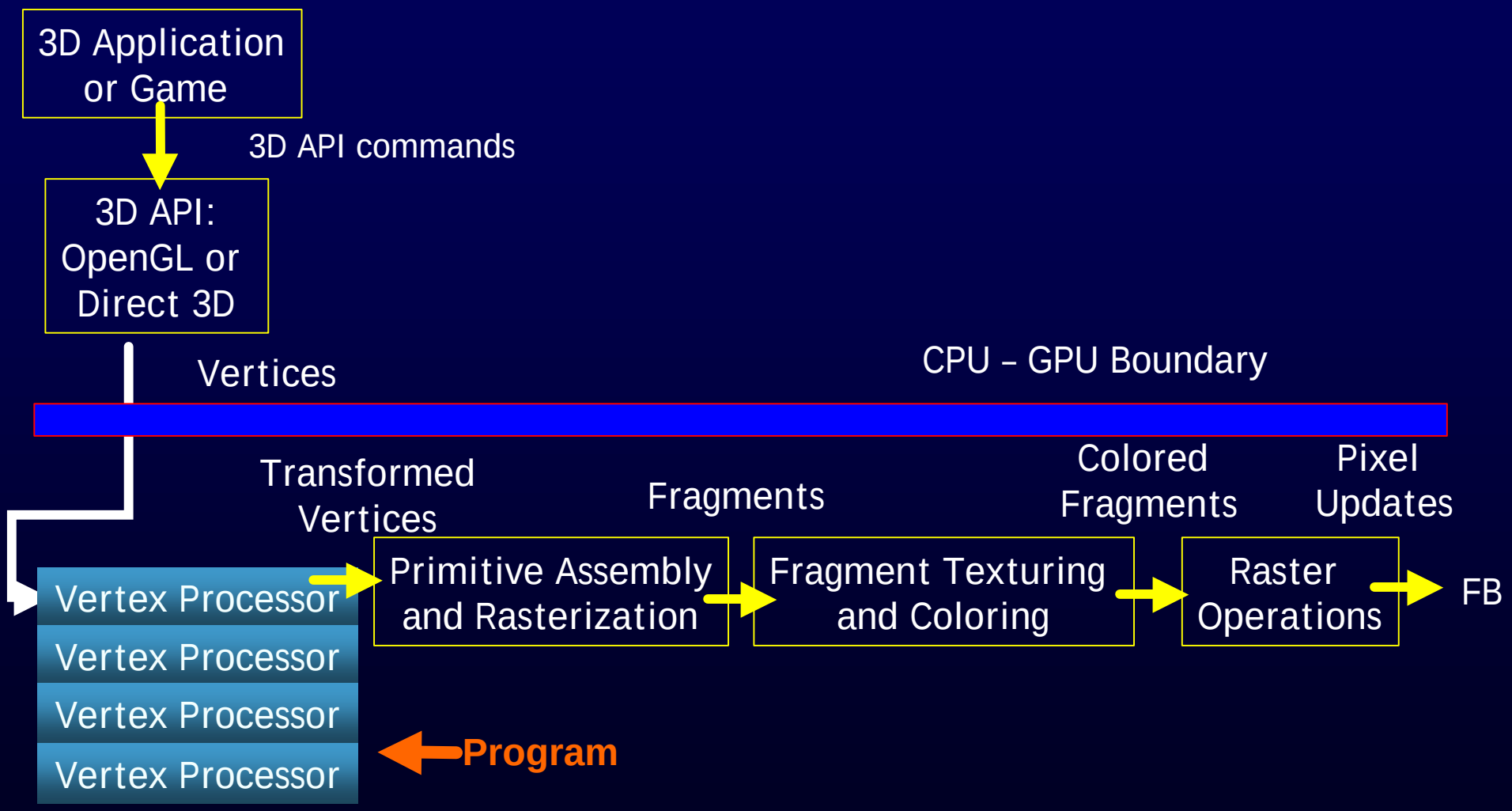
GPU is a stream processor

- ***Multiple programmable processing units***
- ***Connected by data flows***



Programmable Graphics Pipeline

Joao Comba, UFRGS

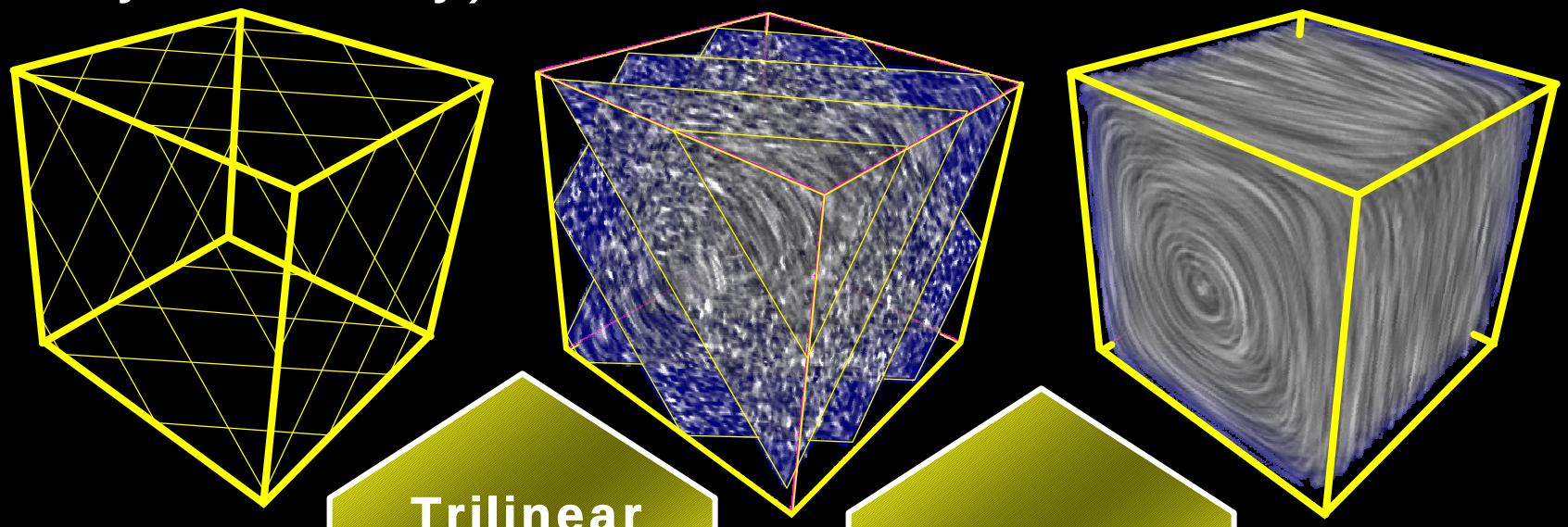


Texture-Based Volume Rendering

Slice Decomposition
(Proxy Geometry)

Rendering of textures slices

Final Image



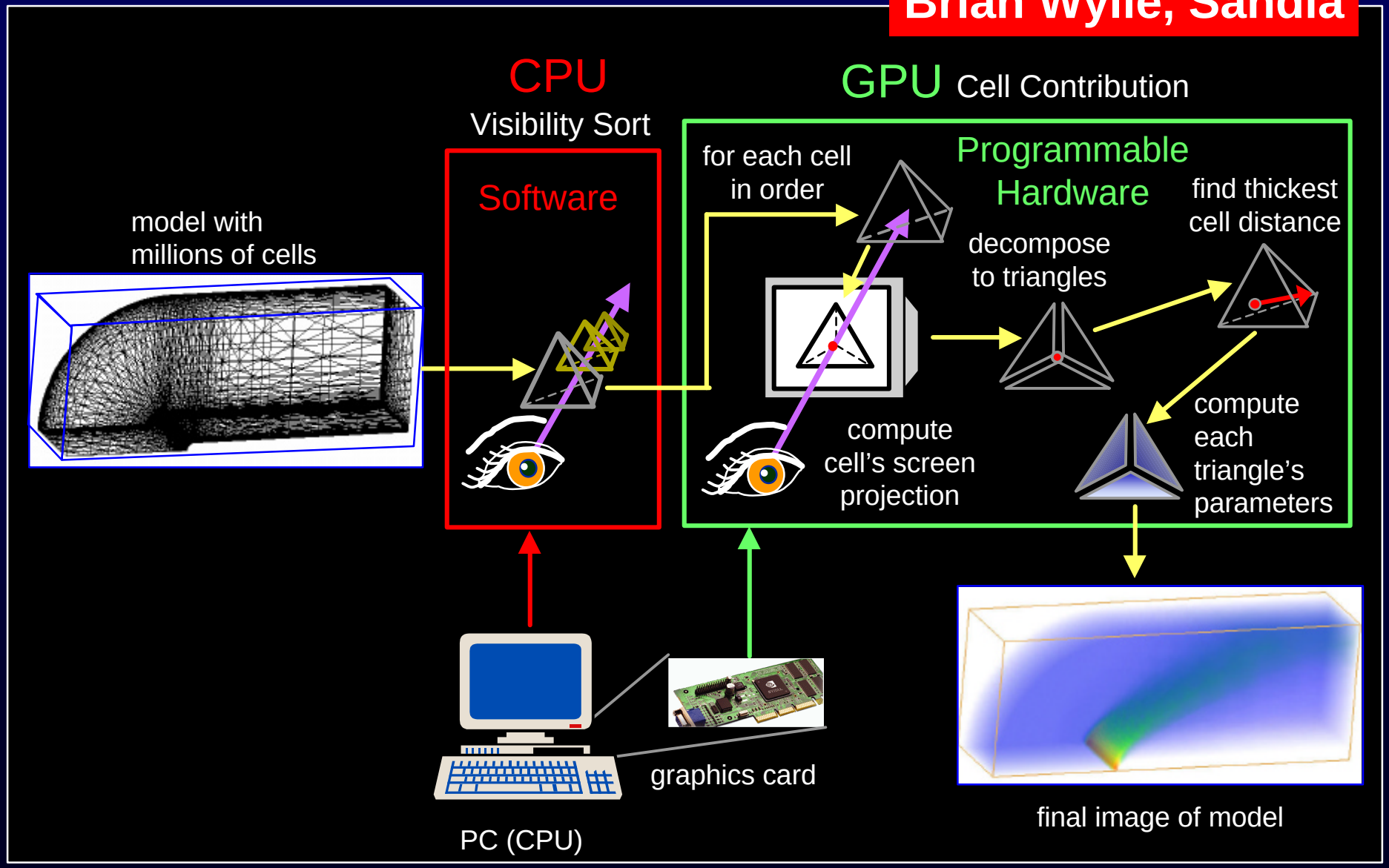
Trilinear Hardware Interpolation

Compositing (Blending)

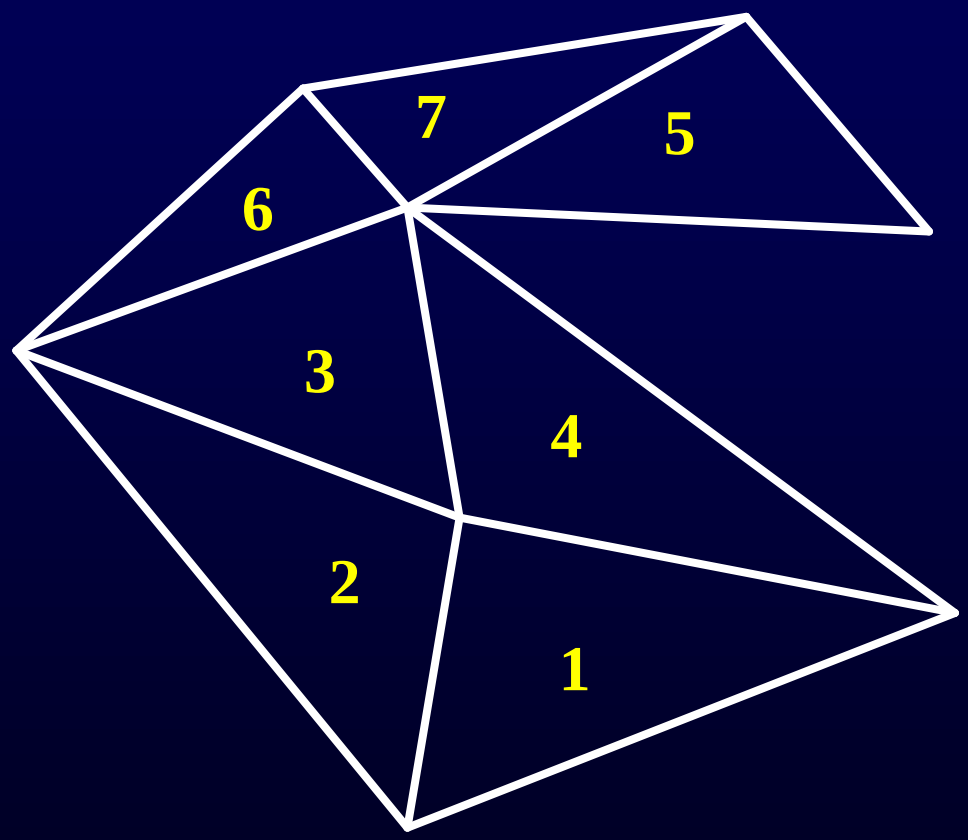
Rezk-Salama Visualization 2002 Tutorial

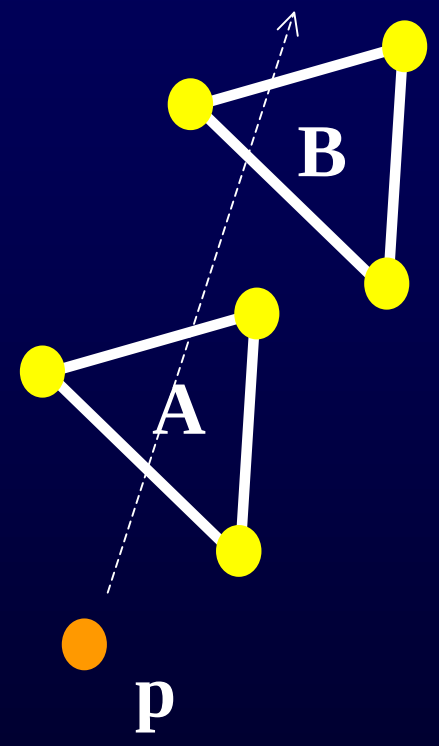
Rendering Unstructured Grids

Brian Wylie, Sandia



Visibility Sorting

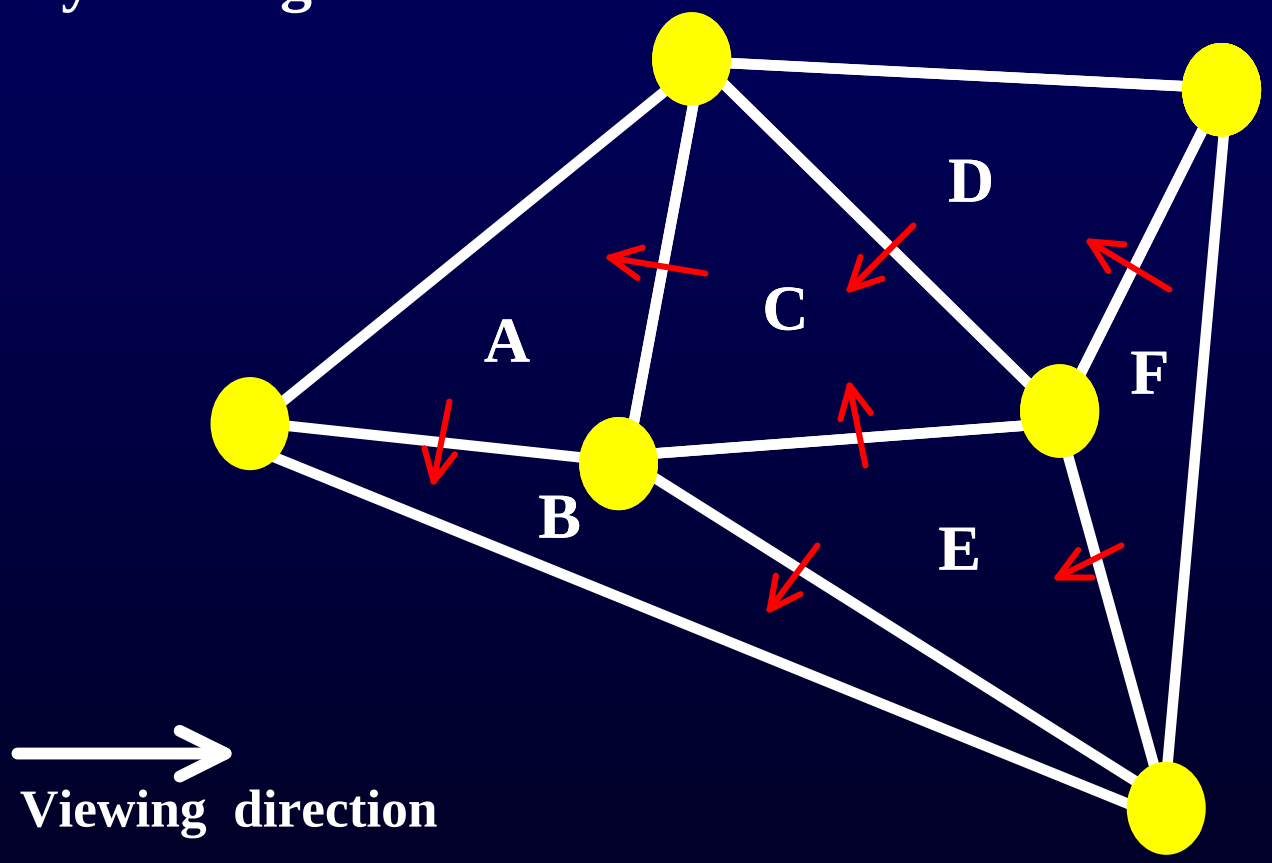




$A <_p B$

Williams' MPVO

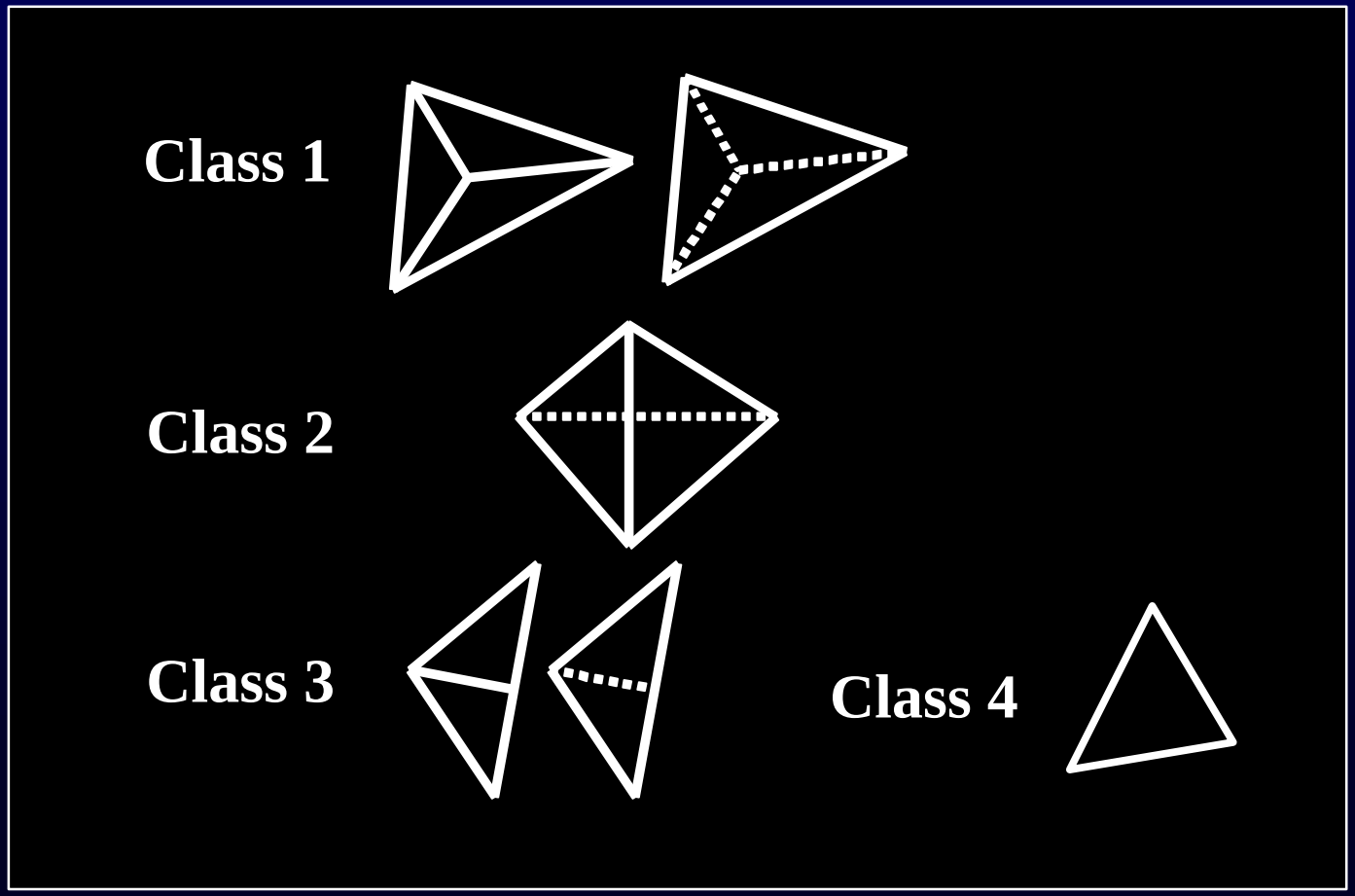
Idea: Define ordering relations by looking at shared faces.



B < A
A < C
B < E
C < E
C < D
E < F
D < F

Shirley-Tuchman (ST) Algorithm

Brian Wylie, Sandia



Wylie et al's GPU-based ST

Brian Wylie, Sandia

Moves all of the following functions from the CPU the GPU:

- Transform to screen space
- Determine projection class
- Calculate thick vertex location
- Determine depth at thick vertex
- Compute color and opacity for thick vertex
- Apply exponential attenuation texture

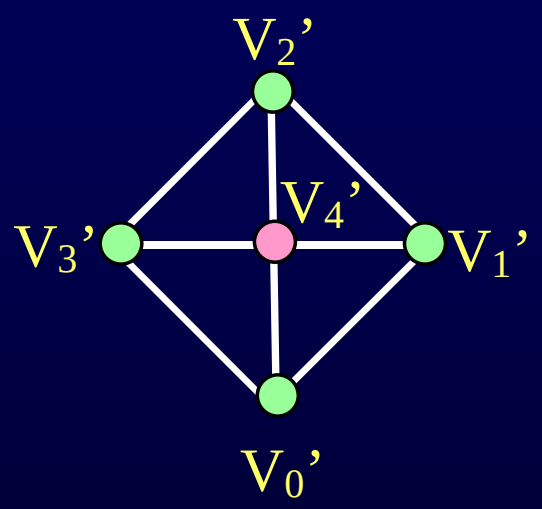
GPU Limitations

Brian Wylie, Sandia

- Each instance of a *vertex shader* program works independently on a single vertex in SIMD fashion
- No support dynamic vertex creation or topology modification within the vertex program
- No branching
- No knowledge of neighboring vertices
- Cannot change execution based on past information

Idea: Morph a Canonical Graph

Brian Wylie, Sandia



Basis Graph

Isomorphic to all projection cases

Example later…

PT algorithm in Vertex Program

Brian Wylie, Sandia

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

PT algorithm in Vertex Program

Brian Wylie, Sandia

- Transform to screen space. (Trivial)
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

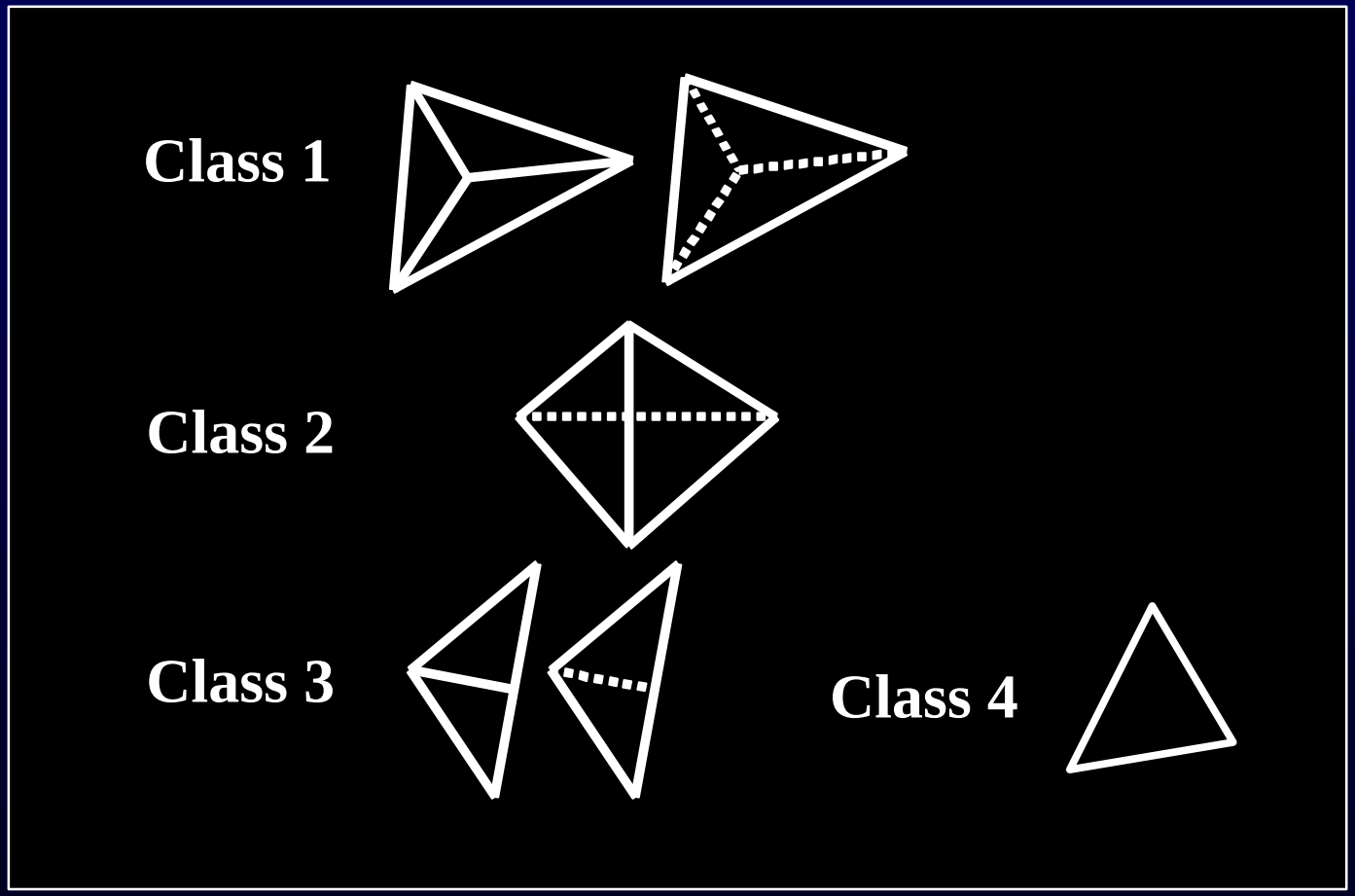
PT algorithm in Vertex Program

Brian Wylie, Sandia

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

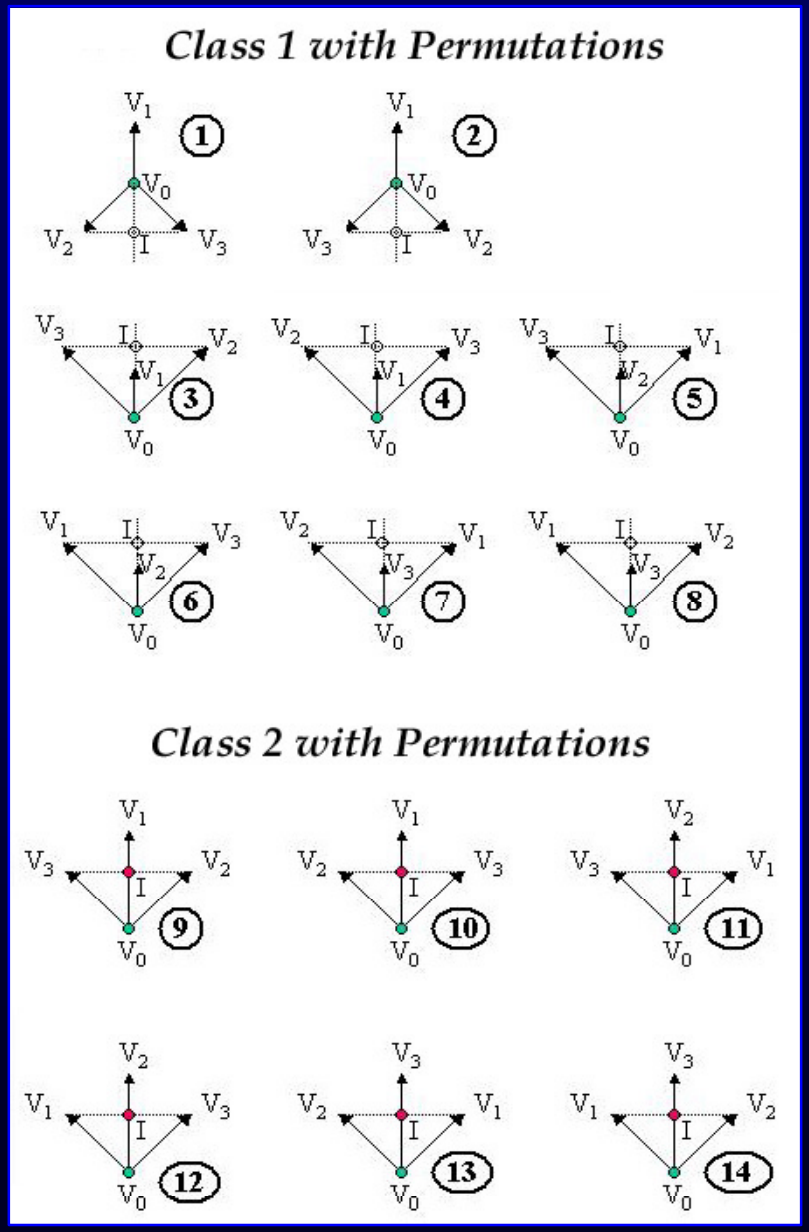
Projection Classes

Brian Wylie, Sandia



Projection Permutations

Brian Wylie, Sandia



Permutation Determination

14 cases need at least 4 Boolean tests

Definitions

vec1 = v1-v0
vec2 = v2-v0
vec3 = v3-v0
cross1 = vec1 x vec2
cross2 = vec1 x vec3
cross3 = vec2 x vec3

Tests

test1 = (cross1*cross2 < 0)
test2 = (cross1*cross3 > 0)
test3 = (distance from v0 to middle vertex – distance from v0 to Intersection) > 0
test4 = (cross1 > 0)

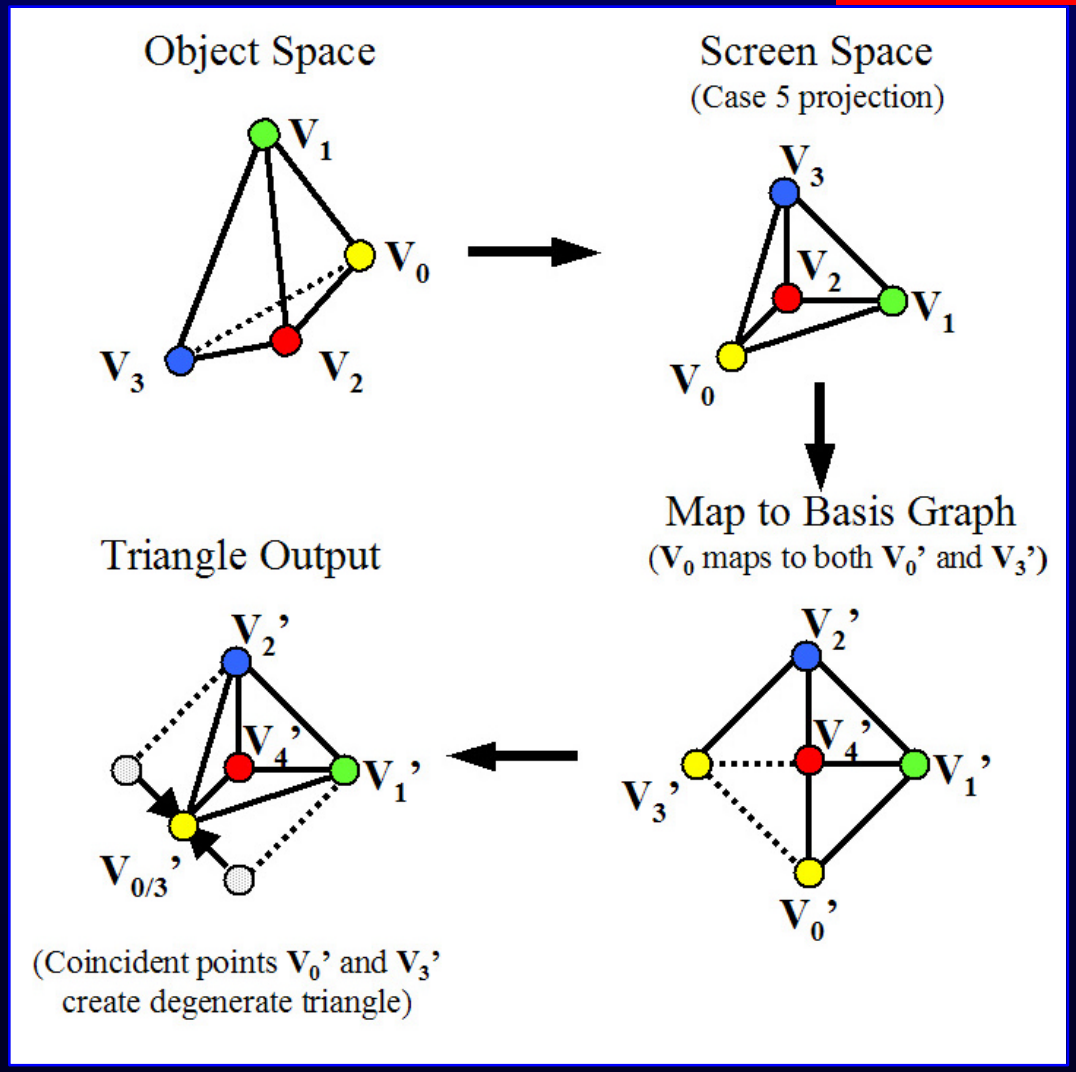
PT algorithm in Vertex Program

Brian Wylie, Sandia

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

Isomorphic Property of Basis Graph

Brian Wylie, Sandia



PT algorithm in Vertex Program

Brian Wylie, Sandia

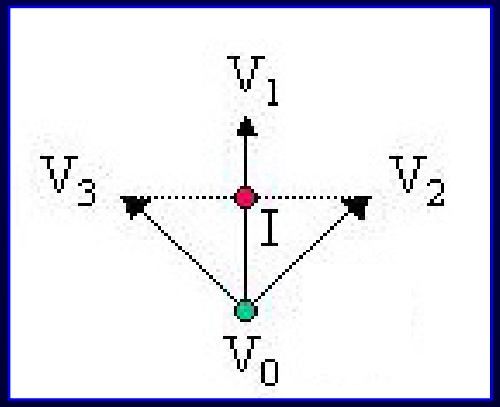
- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

Thick Vertex Calculation

Brian Wylie, Sandia

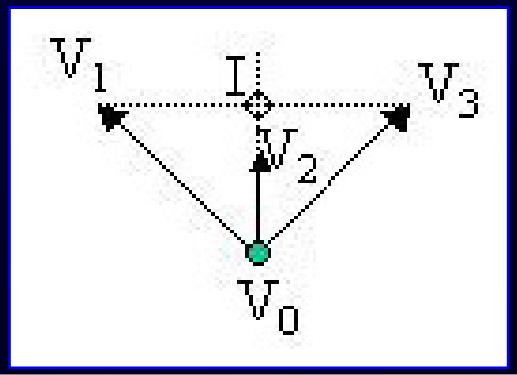
- In all cases the coordinates of the intersection point I are computed. (Intersection of lines computed ala Graphics Gems III p. 199-202). This intersection calculation gives us α and β terms that are used for interpolation (depth, alpha, and color) later on.

- Class 2:




```
// Compute thick vertex "thickness"
float z1 = P1[2] + alpha * (P2[2] - P1[2]);
float z2 = P3[2] + beta  * (P4[2] - P3[2]);
float thickness = fabs(z1-z2);
```

- Class 1:




```
// Extra computation for class 1
if (!test3) thickness /= alpha;
```

PT algorithm in Vertex Program

Brian Wylie, Sandia

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

Color and Opacity Calculation

Brian Wylie, Sandia

- Use same α and β terms to interpolate color and opacity along the line segments to give the front and back face intersection terms C_F, C_B and τ_F, τ_B.
- Thick vertex color $(C_F + C_B) / 2$ *
- The extinction coefficient τ is $(\tau_F + \tau_B) / 2$.
- τ and the thickness l, are then used as lookups into a 2D texture map defined as $1 - \exp(-\tau l)$. [Stein et al. 1994].

* approximate color from Shirley and Tuchman.

PT algorithm in Vertex Program

Brian Wylie, Sandia

- Transform to screen space.
- Determine projection class (and permutation).
- Map the vertices to the *basis graph*.
- Calculate intersection point of line segments.
- Determine depth at thick vertex.
- Compute color and opacity for thick vertex (texture)
- Multiplex the result to correct output vertex.

Multiplex input to output

Brian Wylie, Sandia

Use a lookup table (loaded in the parameter registers) and an index based on the 4 tests to determine the output vertex.

```
// Which vertex to copy to output (using lookup table)
lookup_index = test1*8 + test2*4 + test3*2 + test4;
output_vertex = lookup_table[call_index][lookup_index];
```

‘Feeding’ the Vertex Program

Brian Wylie, Sandia

```
// Load up the 4 vertices
glVertexAttrib3fvNV(1, nodes[0]->getXYZ());
glVertexAttrib3fvNV(2, nodes[1]->getXYZ());
glVertexAttrib3fvNV(3, nodes[2]->getXYZ());
glVertexAttrib3fvNV(4, nodes[3]->getXYZ());

// Load up color for the vertices
glVertexAttrib4fvNV(5, colorvectors[0]);
glVertexAttrib4fvNV(6, colorvectors[1]);
glVertexAttrib4fvNV(7, colorvectors[2]);
glVertexAttrib4fvNV(8, colorvectors[3]);

// Writing to v[0] here invokes the vertex program.
glBegin(GL_TRIANGLE_FAN);
        glVertexAttrib3sNV(0, 0, 1, 0);
        glVertexAttrib3sNV(0, 1, 0, 1);
        glVertexAttrib3sNV(0, 2, 0, 1);
        glVertexAttrib3sNV(0, 3, 0, 1);
        glVertexAttrib3sNV(0, 4, 0, 1);
        glVertexAttrib3sNV(0, 1, 0, 1);
glEnd();
```

These calls could easily be wrapped up into a glTetraExt () call.

Remarks

Wylie et al's technique can be easily extended to other computations, e.g., isosurface or isoline generation (this was a homework exercise in my graphics class last Spring)

For isosurfaces, one can send two triangles (four vertices in a strip), since for a tetrahedral cell, the isosurface going through it has <u>at most two triangles</u>

One student (Francis Chang) found a clever solution with 30 instructions!

Acknowledgements

- DOE VIEWS, DOE MICS, SNL, and LLNL
- National Science Foundation
  - CCR-0306530 and EIA-0323604
- Joao Comba, Ricardo Farias, Brian Wylie, and others for slides and help with presentation