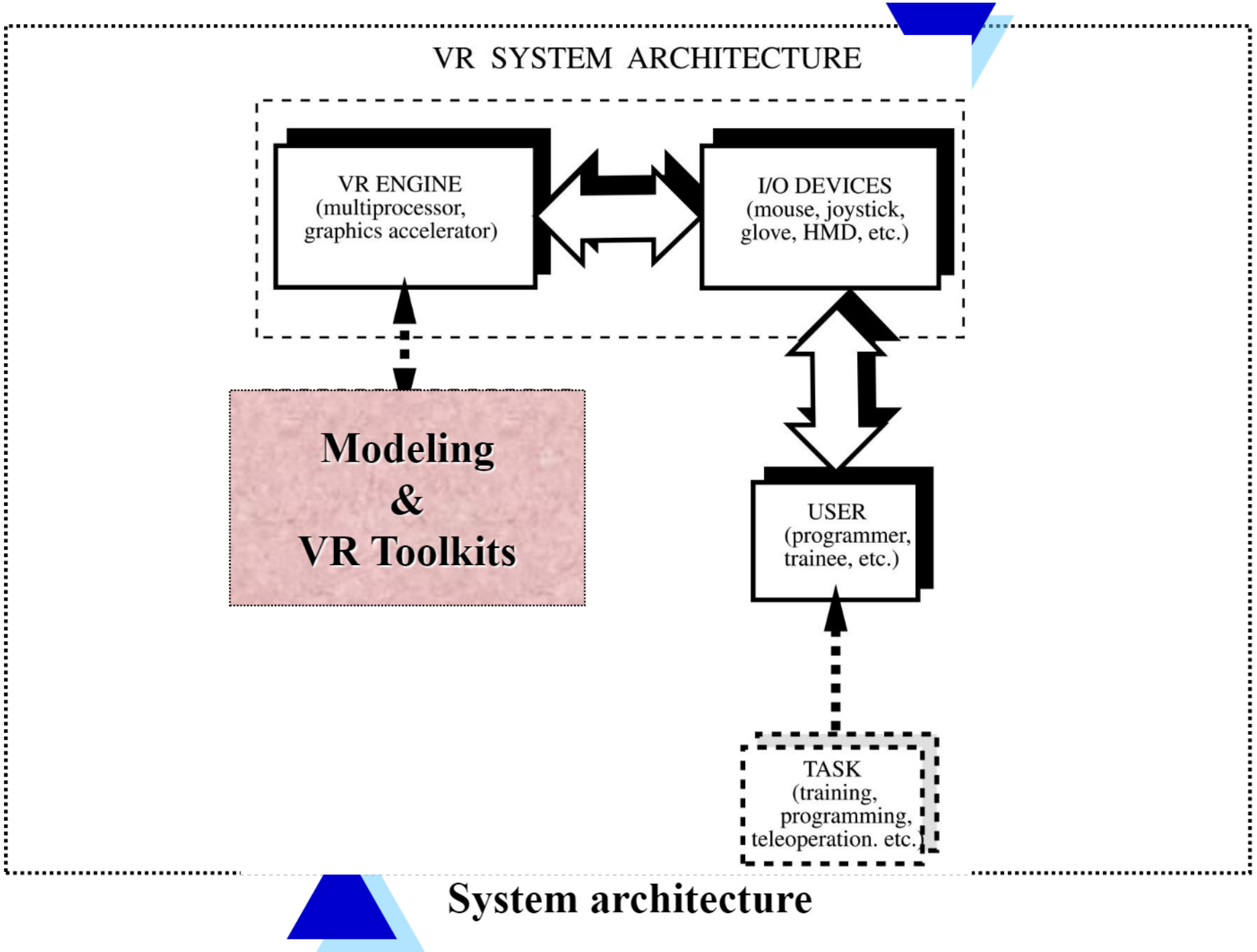
Virtual Reality Modeling

from http://www.okino.com/

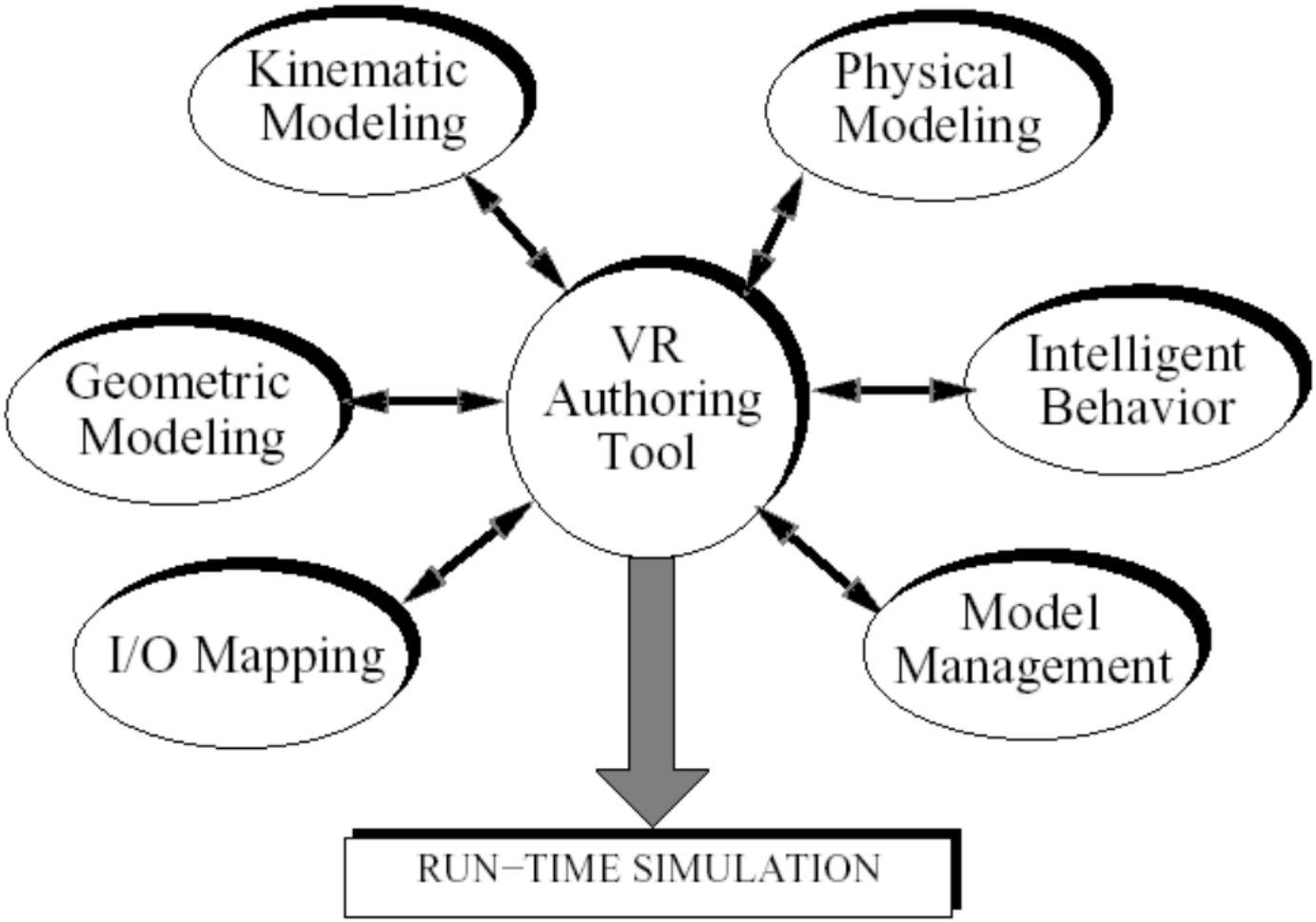


System architecture

The VR object modeling cycle:

- ✓ I/O mapping (drivers);
- ✓ Geometric modeling;
- ✓ Kinematics modeling;
- ✓ Physical modeling;
- ✓ Object behavior (intelligent agents);
- ✓ Model management.

The VR modeling cycle

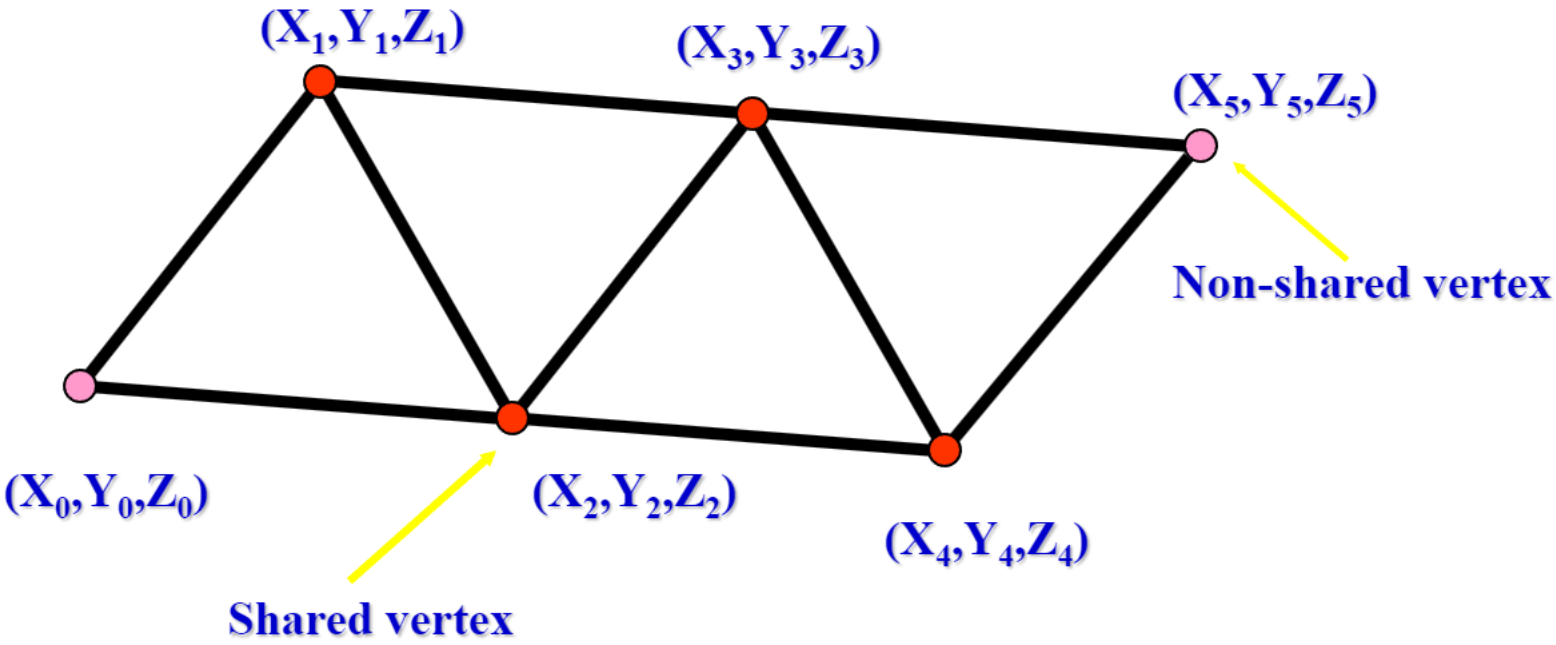


The VR geometric modeling:

- ✓ Object surface shape:
  - polygonal meshes (vast majority);
  - splines (for curved surfaces);

- ✓ Object appearance:
- ✓Lighting (shading)
  - texture mapping

The surface polygonal (triangle) mesh



Triangle meshes are preferred since they are memory and computationally efficient (shared vertices)

Object spline-based shape:

- ✓ Another way of representing virtual objects;
- ✓ Functions are of higher degree than linear functions describing a polygon – use less storage and provide increased surface smoothness.
- ✓ Parametric splines are represented by points x(t), y(t), z(t), t=[0,1] and a, b, c are constant coefficients.

$$x(t) = a_x \cdot t^3 + b_x \cdot t^2 + c_x \cdot t + d_x,$$

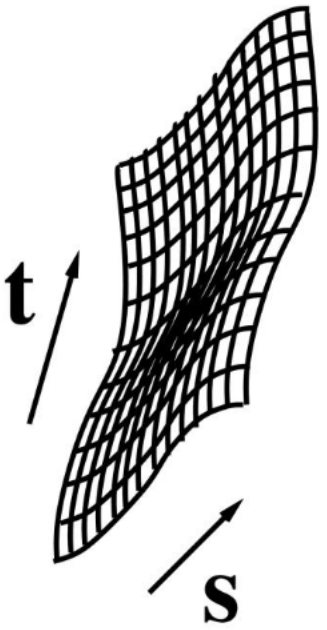
$$y(t) = a_y \cdot t^3 + b_y \cdot t^2 + c_y \cdot t + d_y,$$

$$z(t) = a_z \cdot t^3 + b_z \cdot t^2 + c_z \cdot t + d_z,$$

Object spline-based shape:

✓ Parametric surfaces are extension of parametric splines with point coordinates given by x(s,t), y(s,t), z(s,t), with s=[0,1] and t=[0,1].

β-Splines are controlled indirectly through four control points (more in physical modeling section)



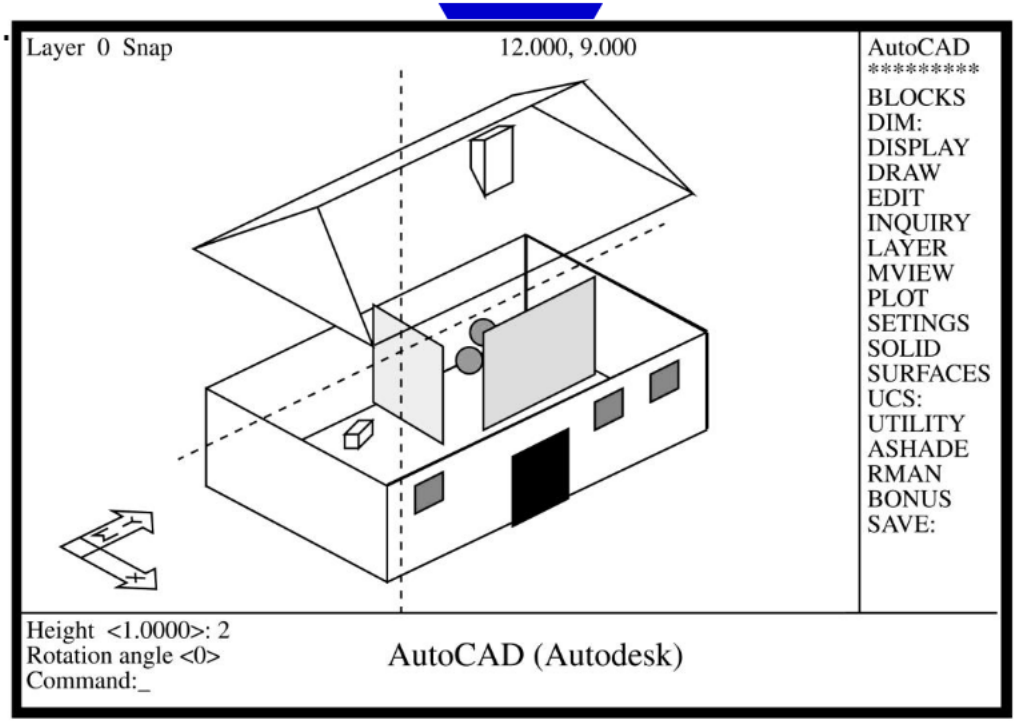
Object polygonal shape:

- ✓ Can be programmed from scratch using OpenGL or other toolkit editor; it is tedious and requires skill;
- ✓ Can be obtained from CAD files;
- ✓ Can be created using a 3-D digitizer (stylus), or a 3-D scanner (tracker, cameras and laser);
- ✓ Can be purchased from existing online databases (Viewpoint database). Files have vertex location and connectivity information, but are ***static***.

Geometric Modeling

CAD-file based models:

- ✓ Done using AutoCAD;
- ✓ Each moving part a separate file;
- ✓ Files need to be converted to formats compatible with VR toolkits;
- ✓ Advantage – use of preexisting models in manufacturing applications.



HOUSE.DXF

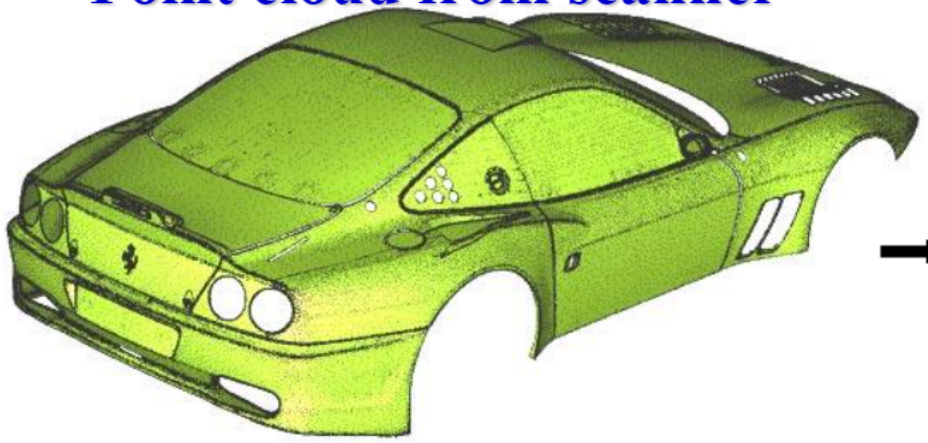
```
LINE
  8
0
  10
2.934
  20
6.5
  30
1.060
  11
4.500
  21
6.500
  31
-0.500
210
0.707
220
0.000
230
0.707
  0
POLYLINE
  8
0
  10
3.292
  20
4.139
  30
0.707
VERTEX
  8
0
  10
4.133
  20
-1.828
  30
2.567
VERTEX
  8
0
  10
4.133
  20
-0.828
  30
2.828
  50
0.000
  0
VERTEX
  8
0
  10
4.1339
  20
.
.
.
.
CIRCLE
  8
0
  10
5.500
  20
2.000
  30
-0.500
  40
0.500
  0
ARC
  8
0
  10
5.560
  20
.
```

Geometric Modeling

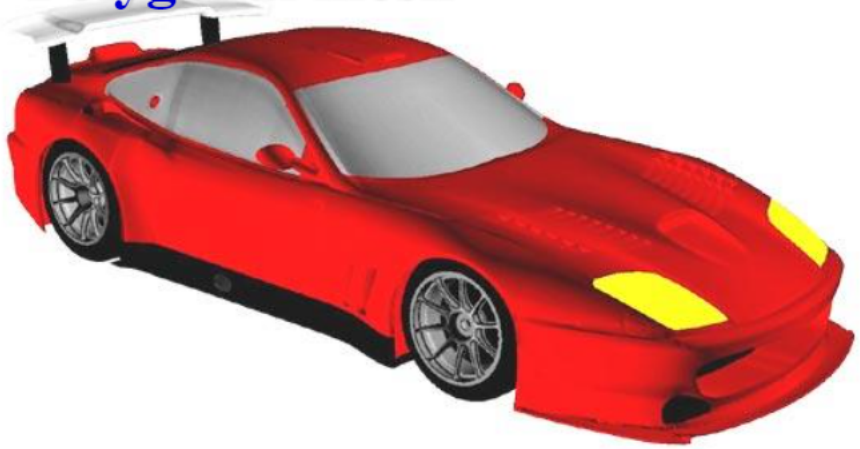
Venus de Milo created using the HyperSpace 3D digitizer, 4200 textured polygons using NuGraph toolkit

Case example: Ferrari down-force rear wing

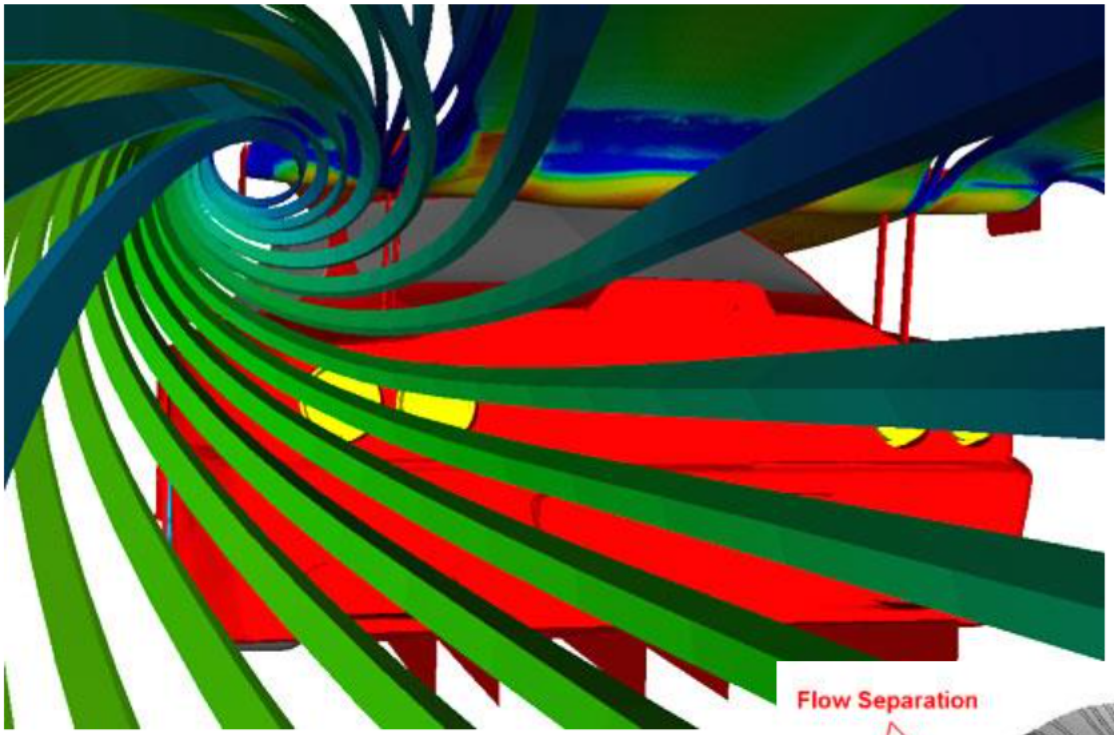
Point cloud from scanner



Polygonal mesh

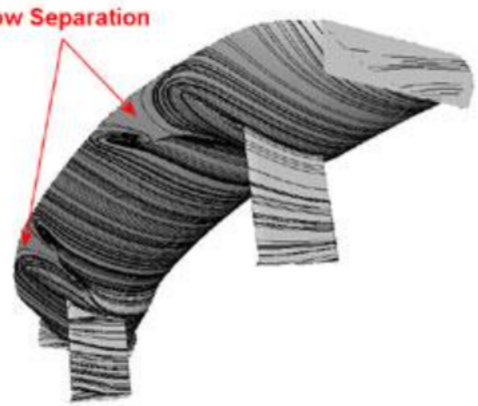


Virtual wind tunnel

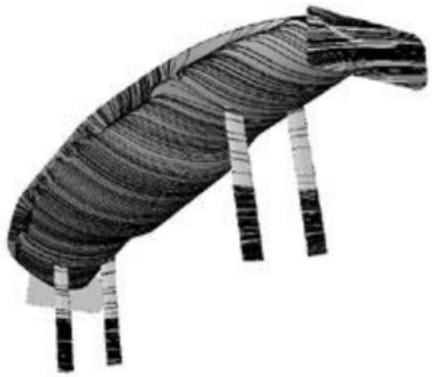


http://www.geomagic.com/en/community/case-studies/prodrive-speeds-ferrari-wing-development-with-reverse-engineerin/

Virtual wind tunnel



PR7 Baseline Wing , Oilflow

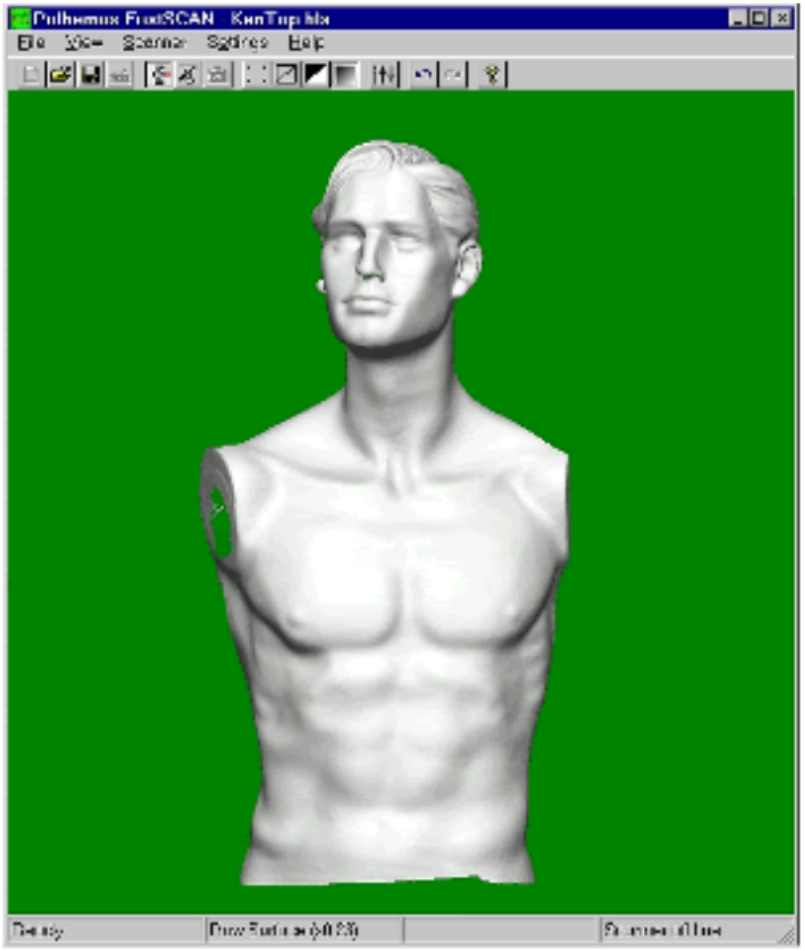


PR7: RUN18 , Oilflow

Polhemus 3-D scanners:

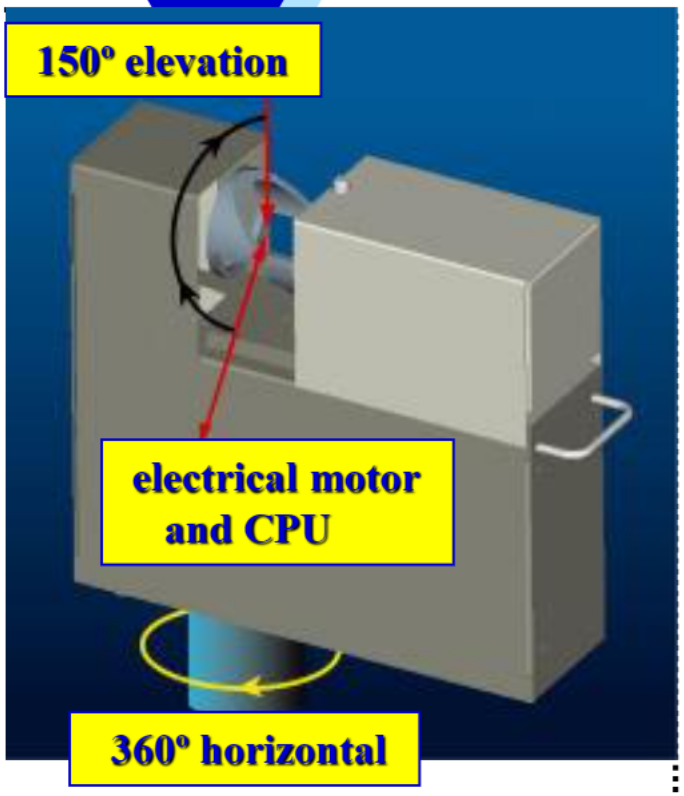
- ✓ Eliminate direct contact with object.
- ✓ uses two cameras, a laser, and magnetic trackers (if movable objects are scanned)
- ✓ Scanning resolution 0.5 mm at 200 mm range;
- ✓ Scanning speed is 50 lines/sec;
- ✓ Range is 75-680 mm scanner-object range.

Geometric Modeling



Polhemus FastScan 3D scanner (can scan objects up to 3 m long).

DeltaSphere 3000 3D scanner



✓ Large models need large-volume Scanners;

✓ The 3rdTech scanners use time-of-flight modulated laser beam to determine position.

Features:

- Scanning range up to 40 ft;
- Resolution 0.01 in;
- accuracy 0.3 in;
- scan density of up to 7200 samples/360º;
- complete scene scanning in 10 – 30 minutes (scene has to be *static*);
- optional digital color camera (2008x1504 resolution) to add color to models. Requires a second scan, and reduces elevation to 77º.

Polhemus scanner

DeltaSphere 3000 3D scanner

Feature	Polhemus scanner	DeltaSphere scanner
Range	0.56 m	14.6 m
Resolution	0.5 mm @ 0.2 m	0.25 mm
Control	manual	automatic
Speed	50 lines/sec	25,000 samples/sec

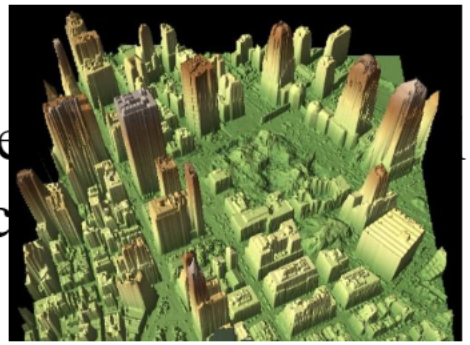
www.3rdtech.com

DeltaSphere 3000 image

DeltaSphere 3000 software-compensated image

Light Detection And Ranging (LIDAR)

- Optical remote sensing technology that can measure the distance to, or other properties of a target by illuminating the target with Light, often using pulses from a laser.



-Geomatics, archaeology, geography, geology, ge
smology, forestry, remote sensing and atmospheric

-Also for 'airborne laser swath mapping' (ALSM), 'laser altimetry' and LiDAR Contour Mapping.

This LiDAR (laser range finder) may be used to scan buildings, rock formations, etc., to produce a 3D model. The LiDAR can aim its laser beam in a wide range: its head rotates horizontally, a mirror flips vertically. The laser beam is used to measure the distance to the first object on its path.

A FASOR used at the Starfire Optical Range for LiDAR and laser guide star experiments is tuned to the sodium D2a line and used to excite sodium atoms in the upper atmosphere.

How it work?

- A laser generates an optical pulse

- The pulse is reflected off an object and is returned to the instrument

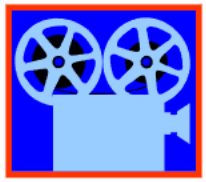
- A high-speed counter measures the time of flight from the start to the return pulse

- Finally, the time measurement is converted to a distance using the formula: $R = (T * Э)/2$ (1)

Where R = range (m)

T = time of flight(s)

Э = speed of light(m/s)

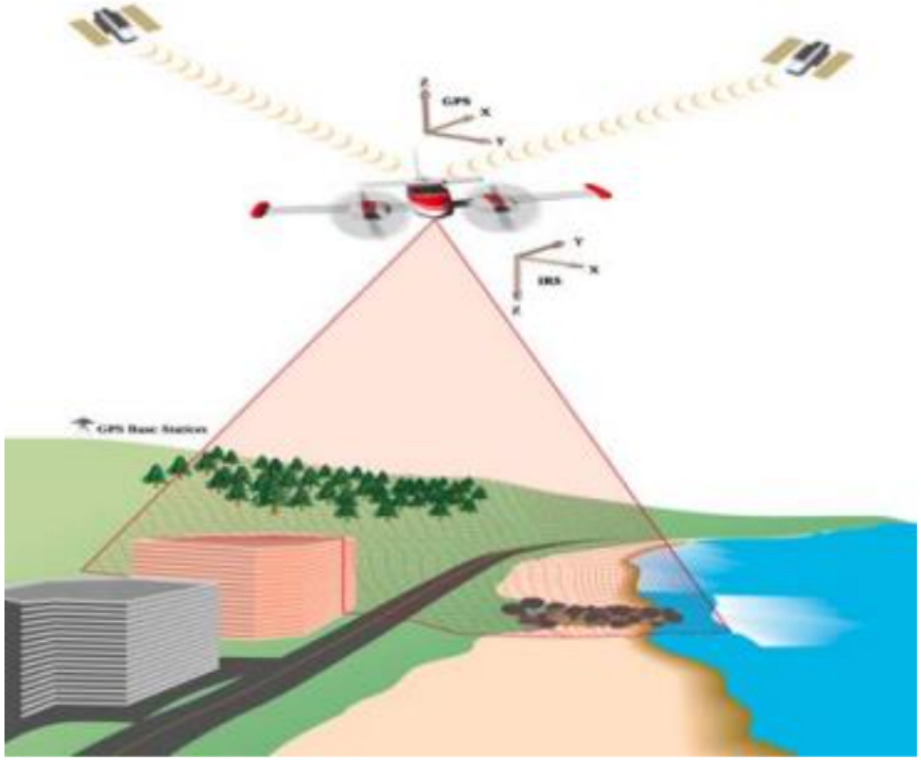


Three mature technologies are combined in airborne laser mapping:

- laser rangefinder (LiDAR)

- Inertial reference system (INS) and

- Global positioning system (GPS).



Hardware

Optech's Airborne Laser Terrain Mapper (ALTM)

The ALTM 2050 collects 50,000 measurements per second and has a maximum altitude of 2000 m.

Maintain balance between scanner speed and aircraft speed.

Optech's ILRIS-3D tripod-mounted laser scanner.

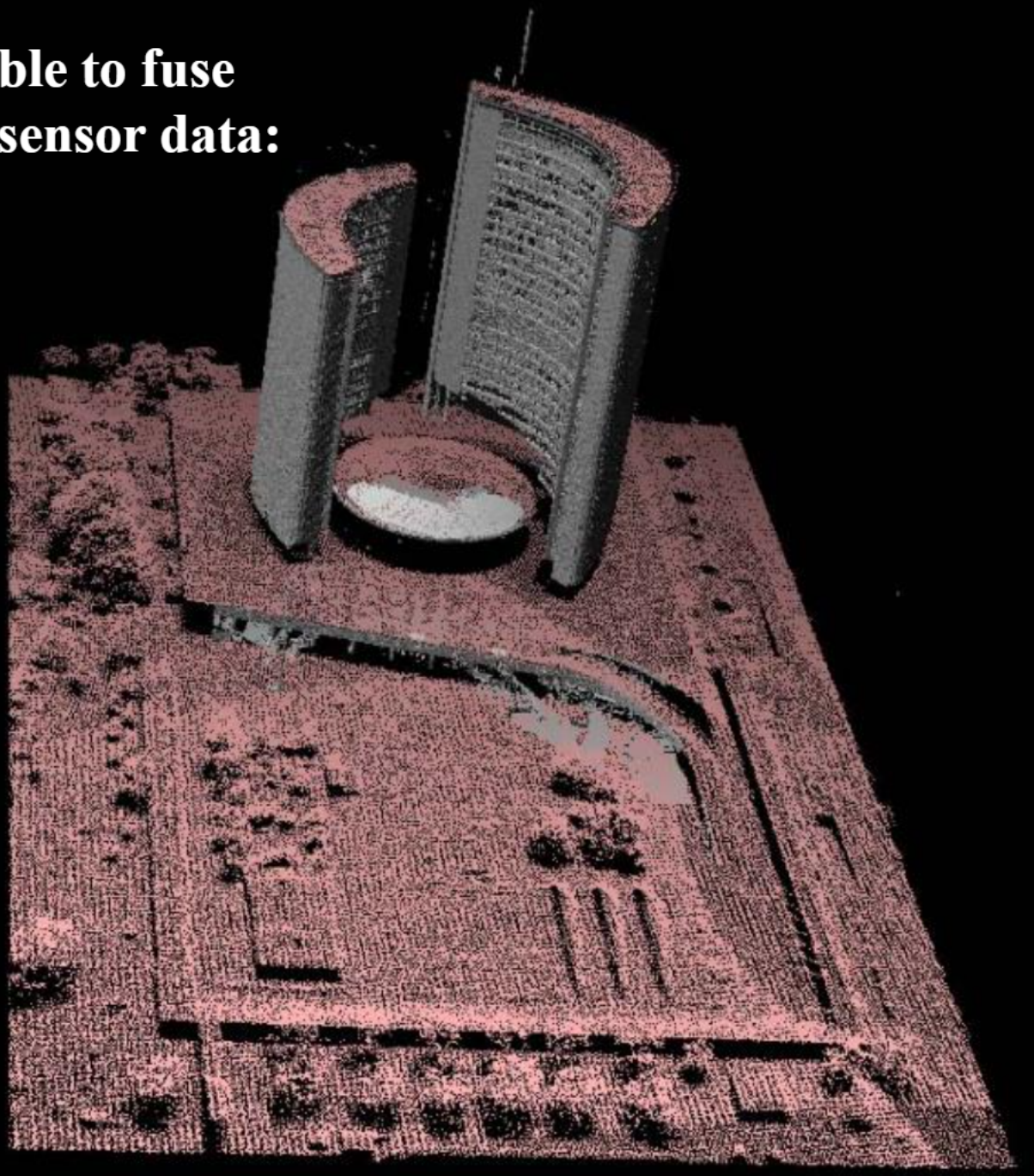
It has long ranging capabilities (up to 1000 m)

Obtaining a line of sight that allowed for economical data collection was challenging

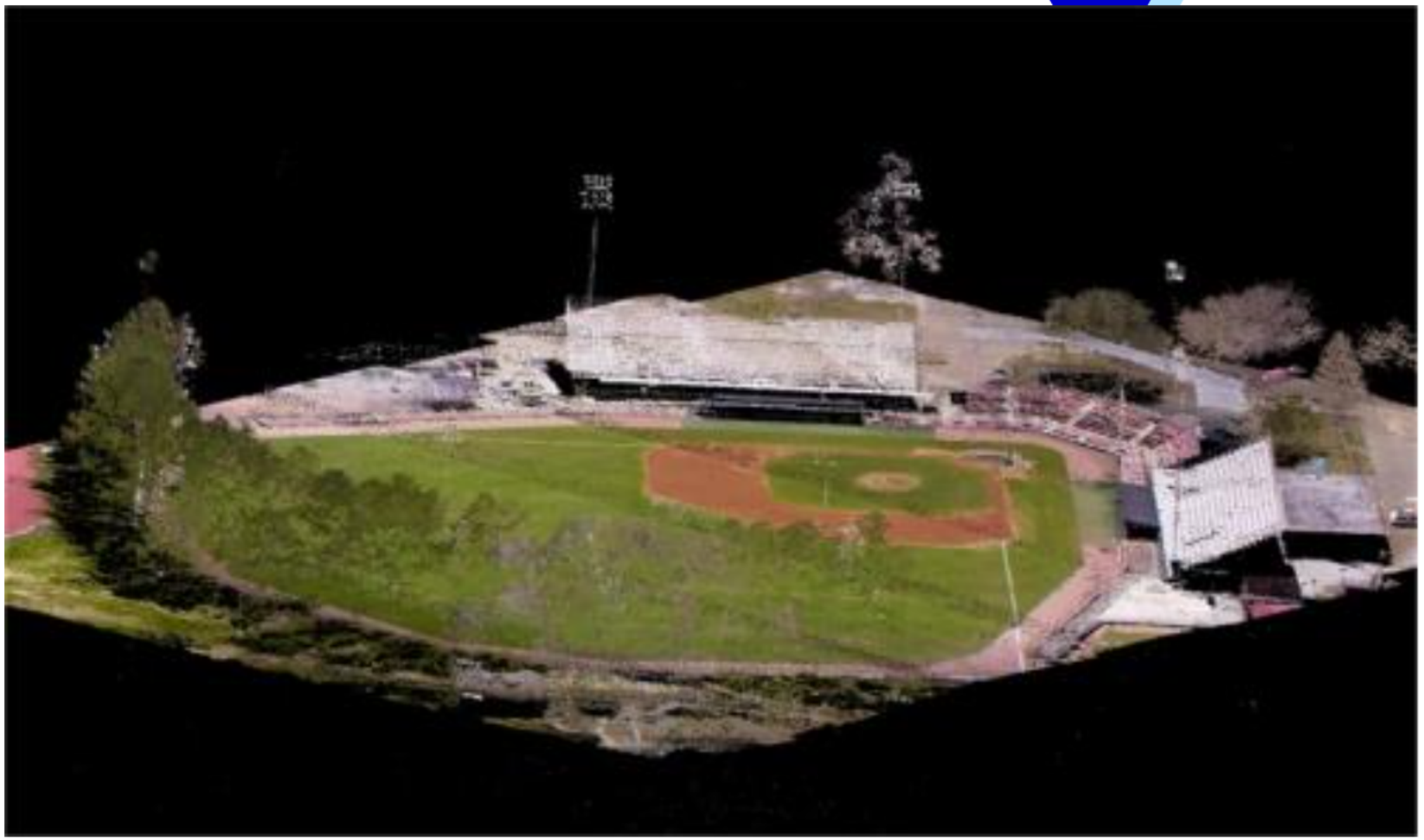
Two methods were available to fuse the ground and airborne sensor data:

-Geo referencing the ground and airborne data, or

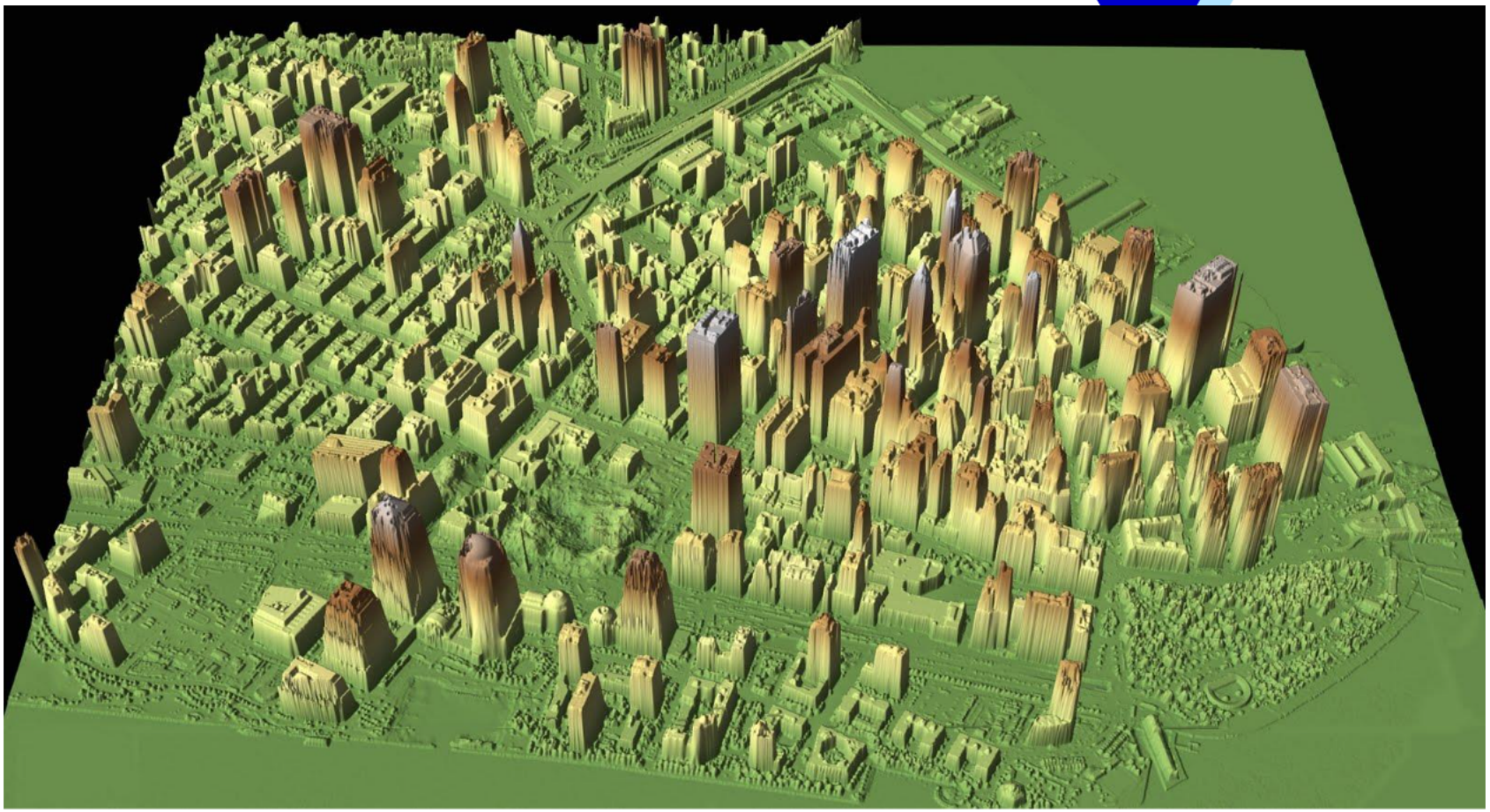
-Aligning both data sets by identifying common points.



Fused sensor data:
ILRIS-3D data is grey,
ALTM data is pink



A LiDAR point cloud captured from above can be rotated and colorized with RGB values from a digital aerial photo to produce a realistic 3D oblique view



This is a LIDAR map of the Manhattan, New York. This LIDAR map was used after 9/11 to determine what cranes are necessary to remove the rubble

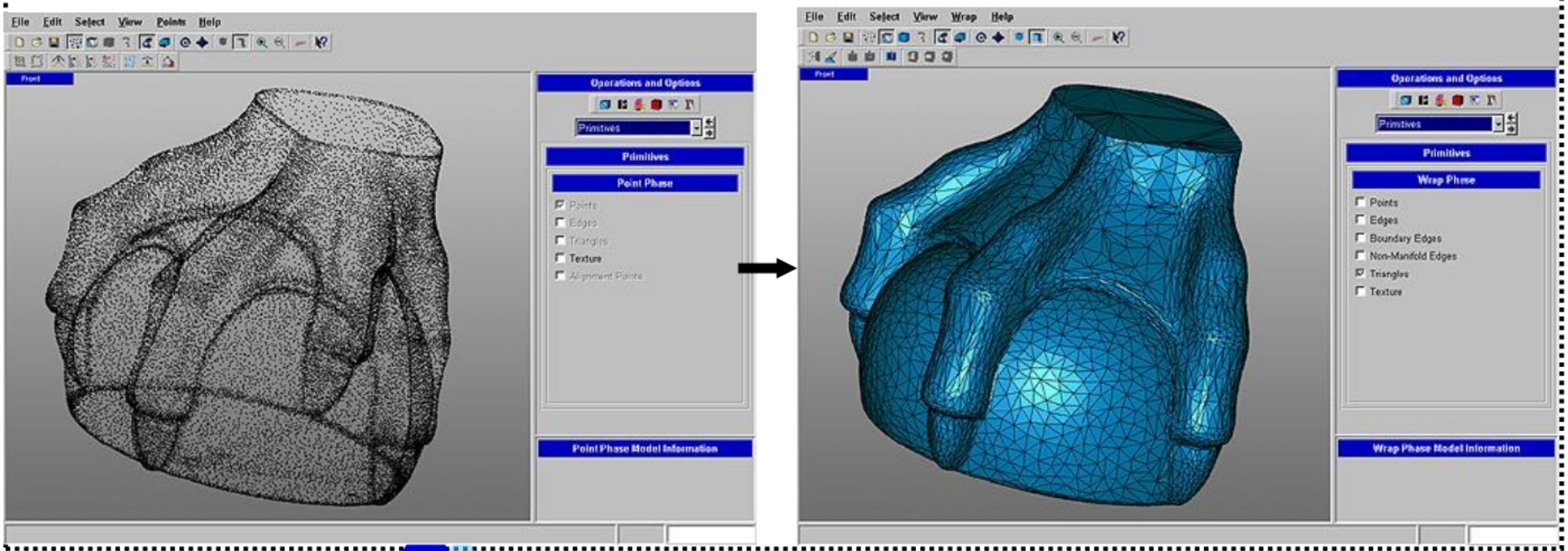
http://www.noaanews.noaa.gov/stories/s798.htm

Conversion of scanner data:

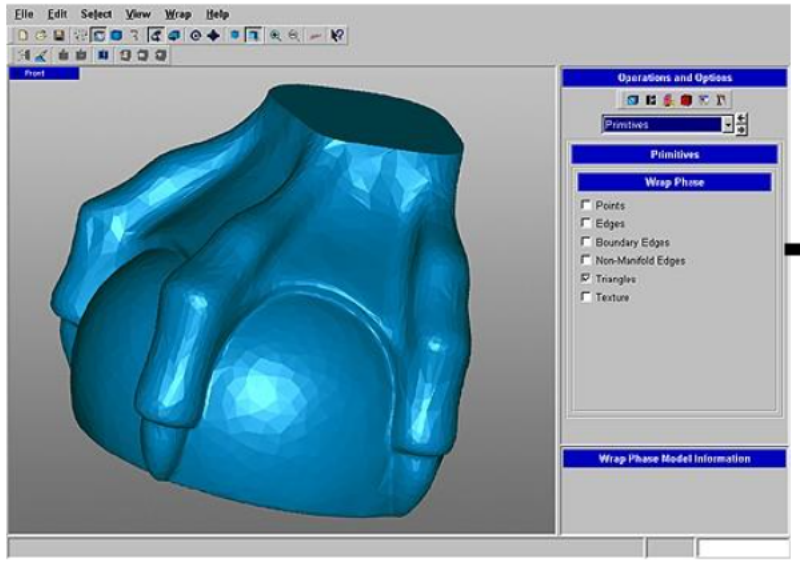
- ✓ Scanners produce a dense "cloud" of vertices (x,y,z).
- ✓ Using such packages as ***Wrap*** (www.geomagic.com) the point data are transformed into surface data (including editing and decimation)

Point cloud from scanner

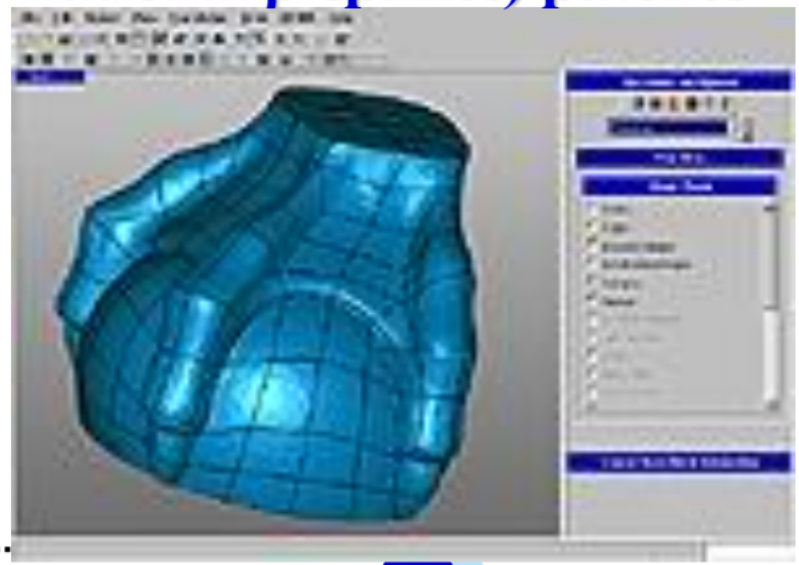
Polygonal mesh after decimation



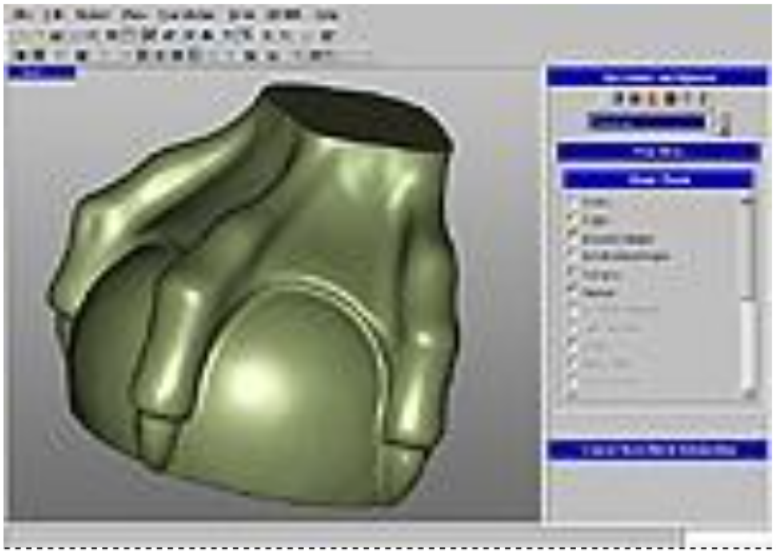
Polygonal surface



NURBS (non-uniform rational β-splines) patches



NURBS surface



Geometric Modeling – using online databases

Low res. Model – 600 polygons

Higher resolution model > 20,000 polygons.

Geometric Modeling

Table 5.1 Methods for modeling 3-D object geometry.

METHOD	FEATURE	SOURCE
Toolkit Editors	Tedious, requires skill	OpenGL, Starbase, PHIGS
CAD Programs	Interactive, existing technology	AutoCAD (Autodesk) 3-D Studio, etc.
3-D Digitizers	Interactive, Allows custom models	Autodesys Inc. Mira Imaging, Polhemus Inc.,etc.
3-D Scanners	Fast multi-point acquisition Large objects	Polhemus Inc., Cyra Technologies, etc.
Commercial 3-D databases	Vertice list, connectivity, static model, level of detail	Viewpoint Inc., etc.

Object Visual Appearance

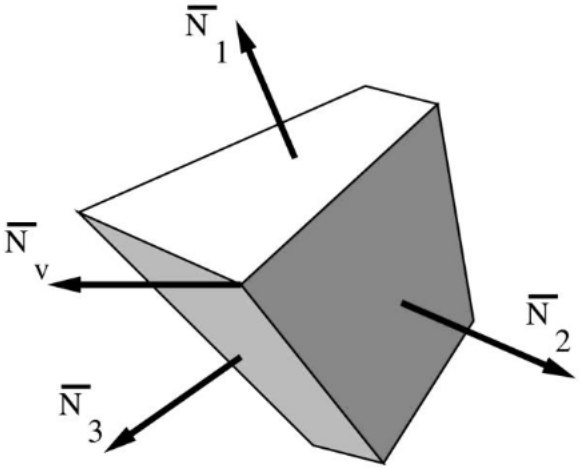
- ✓ Scene illumination (local or global);
- ✓ Texture mapping;
- ✓ Multi-textures
- ✓ Use of textures to do illumination in the rasterizing stage of the pipeline

Scene illumination

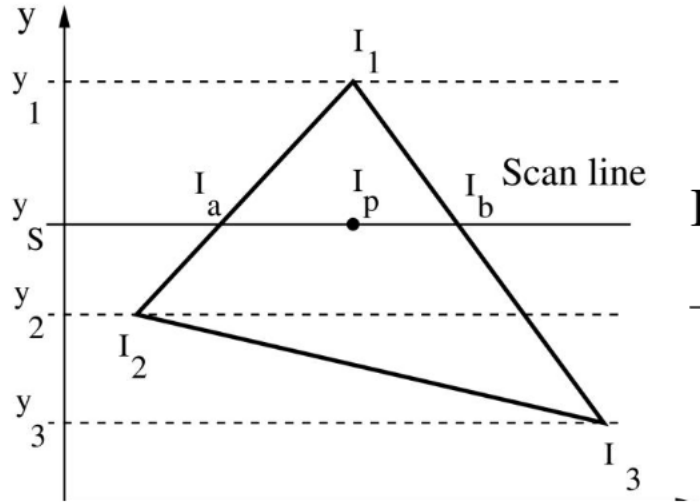
✓ Local methods (Flat shaded, Gouraud shaded, Phong shaded) treat objects in *isolation*. They are computationally faster than global illumination methods;

✓ Global illumination treats the *influence of one object on another* object's appearance. It is more demanding from a computation point of view but produces more realistic scenes.

Local illumination methods

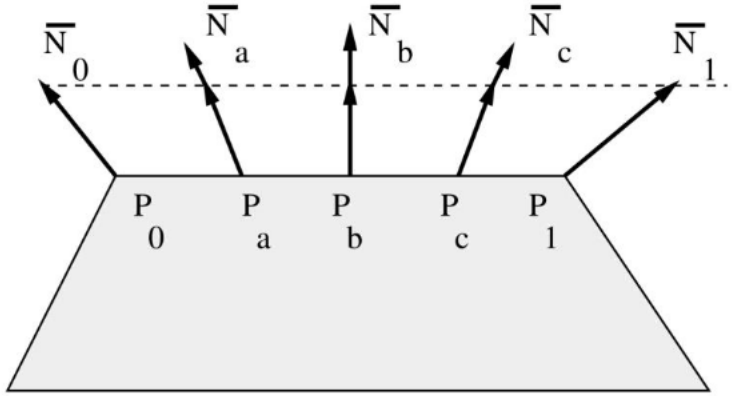


Flat shading model



$$I_p = I_b - (I_b - I_a)\frac{x_b - x_p}{x_b - x_a}$$

Gouraud shading model



Phong shading model

Flat shaded Utah Teapot

Phong shaded Utah Teapot

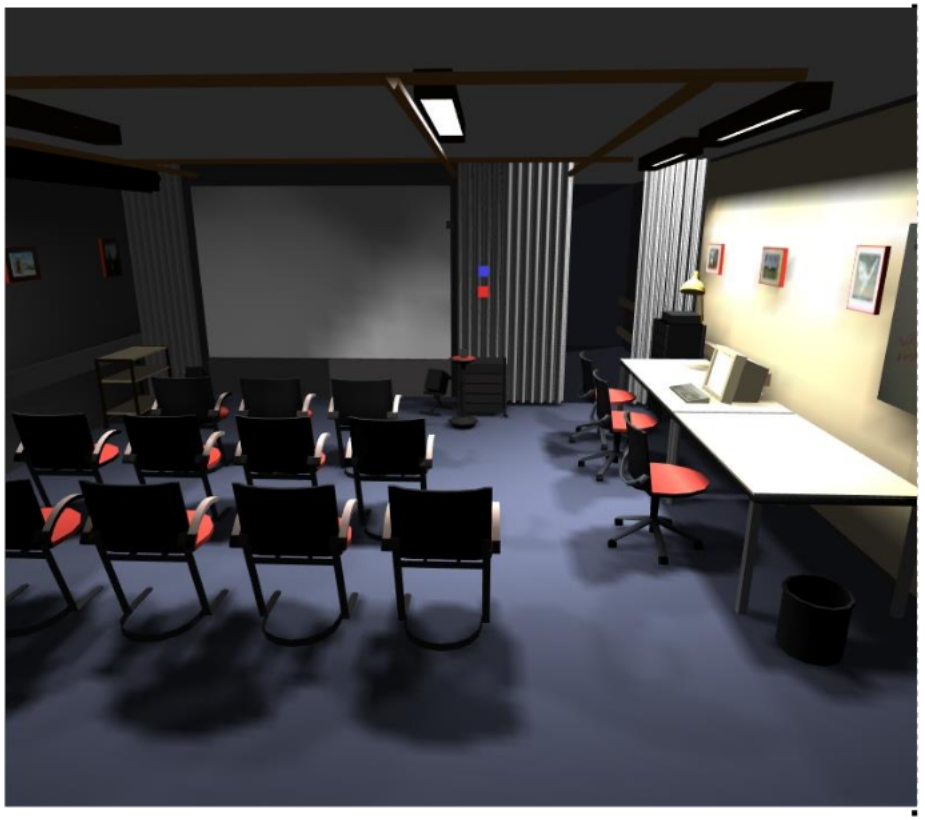
Global scene illumination

✓ The inter-reflections and shadows cast by objects on each other.

Radiosity illumination

- ✓ Results in a more realistic looking scene

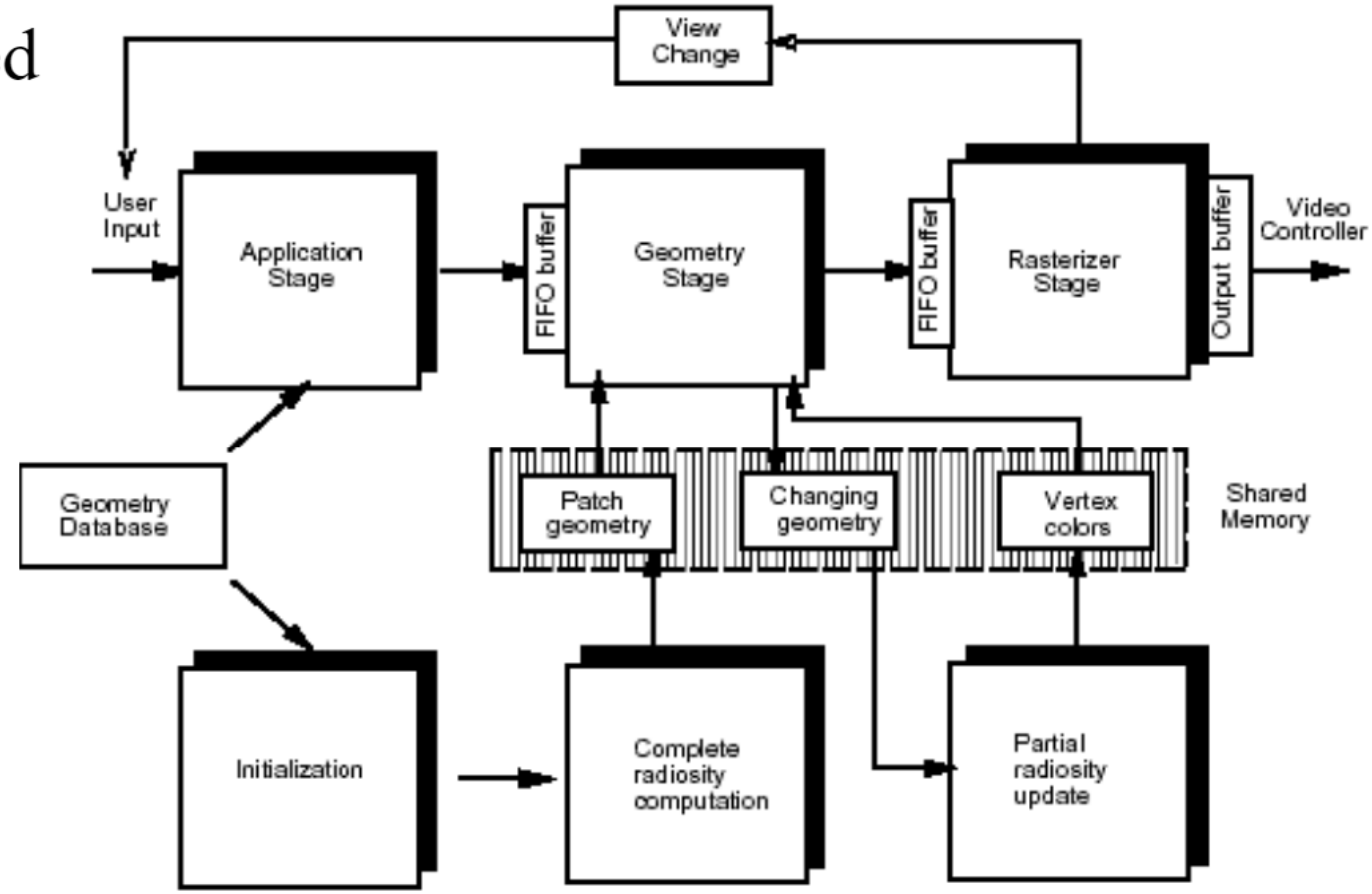
Without radiosity



With radiosity

Radiosity illumination

- ✓… but until recently only for fly-through (geometry fixed).
- ✓A second process was added so that scene geometry can be altered



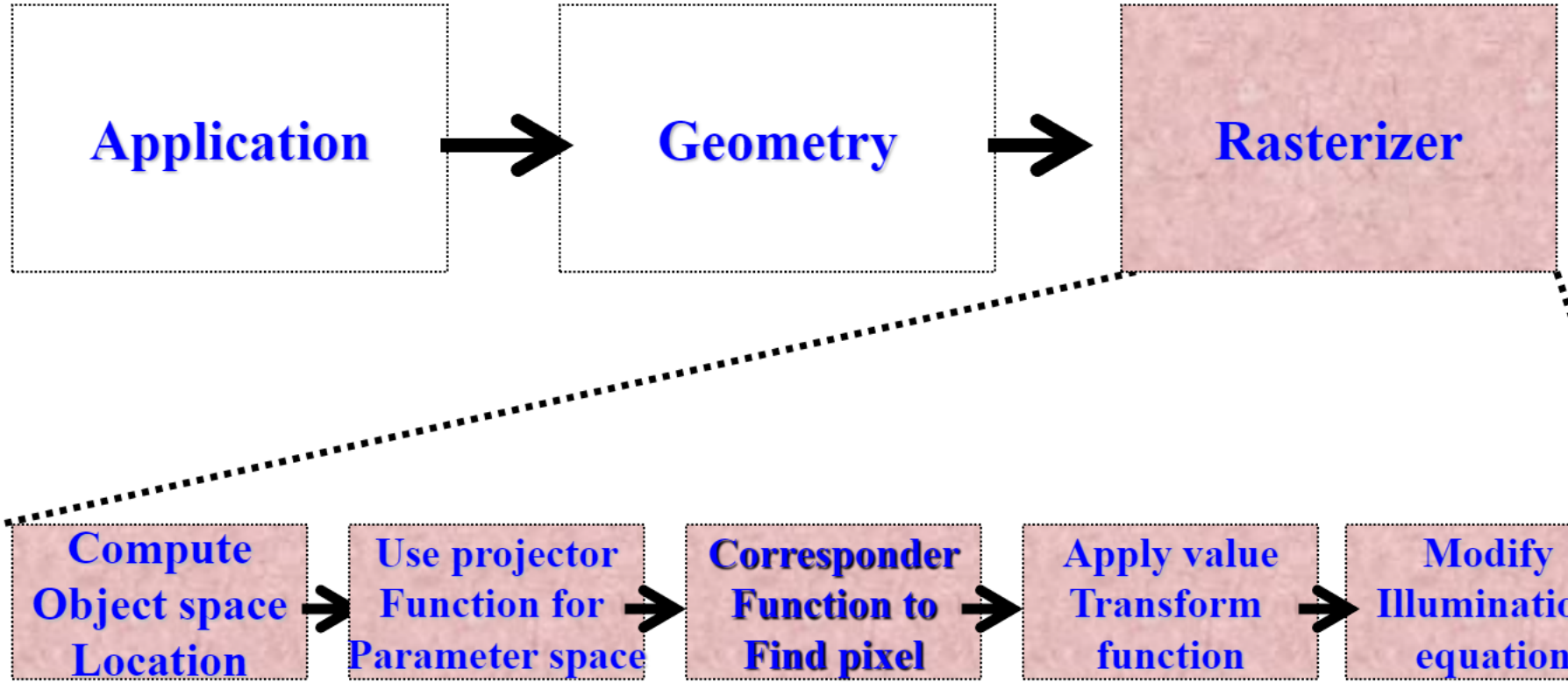
Texture mapping

- ✓ It is done in the rasterizer phase of the graphics pipeline, by mapping texture space coordinates to polygon vertices (or splines), then mapping these to pixel coordinates;
- ✓ Texture ***increase scene realism***;
- ✓ Texture provide better 3-D spatial cues (they are perspective transformed);
- ✓ They reduce the number of polygons in the scene – increased frame rate (example – tree models).

Textured room image for increased realism

from http://www.okino.com/

The Texturing Functional Sub-Stages



How to create textures:

- ✓ Models are available on line in texture "libraries" of cars, people, construction materials, etc.

- ✓ Custom textures from scanned photographs or
- ✓ Using an interactive paint program to create bitmaps

Texture mapping

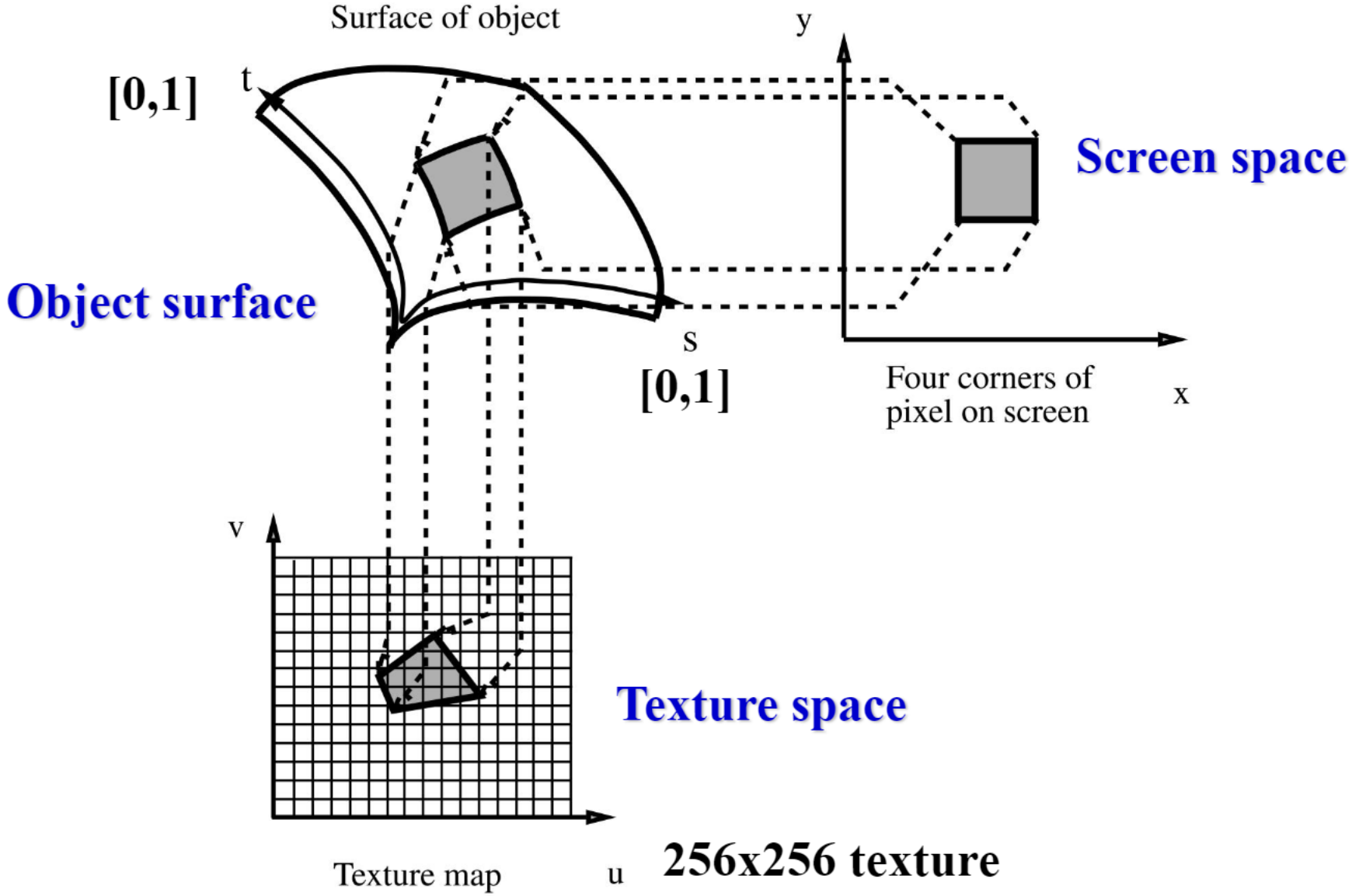


Image Texture:

- ✓ It "glues" an image to a polygon.
- ✓ Size of texture is restricted by graphics accelerators to $2^m \times 2^n$ or $2^m \times 2^m$ square. The lower limit for OpenGL is 64×64 texels.
- ✓ If the size of the polygon is much larger than the size of the texture then the hardware has to perform ***Magnification;***
- ✓ If there are much fewer pixels than texels – ***minification;***
- ✓ Both techniques use bilinear interpolation to assign colors to pixels.

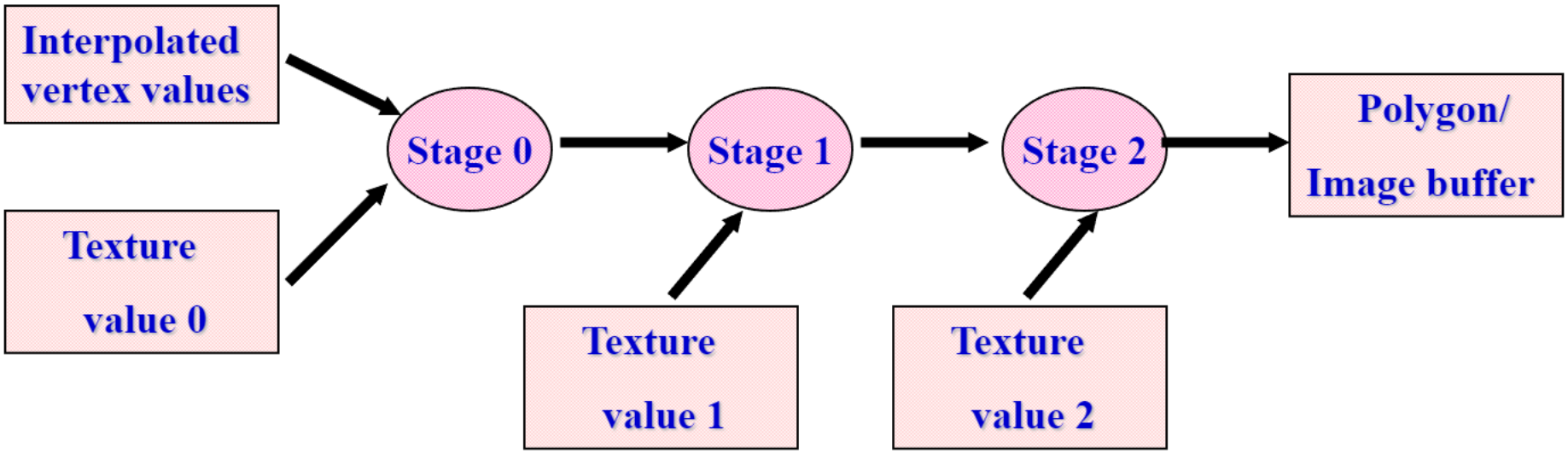
VR Geometric Modeling

Tree,
higher resolution model
45,992 polygons.

Tree represented as a texture
1 polygon, 1246x1280 pixels
(www.imagecels.com).

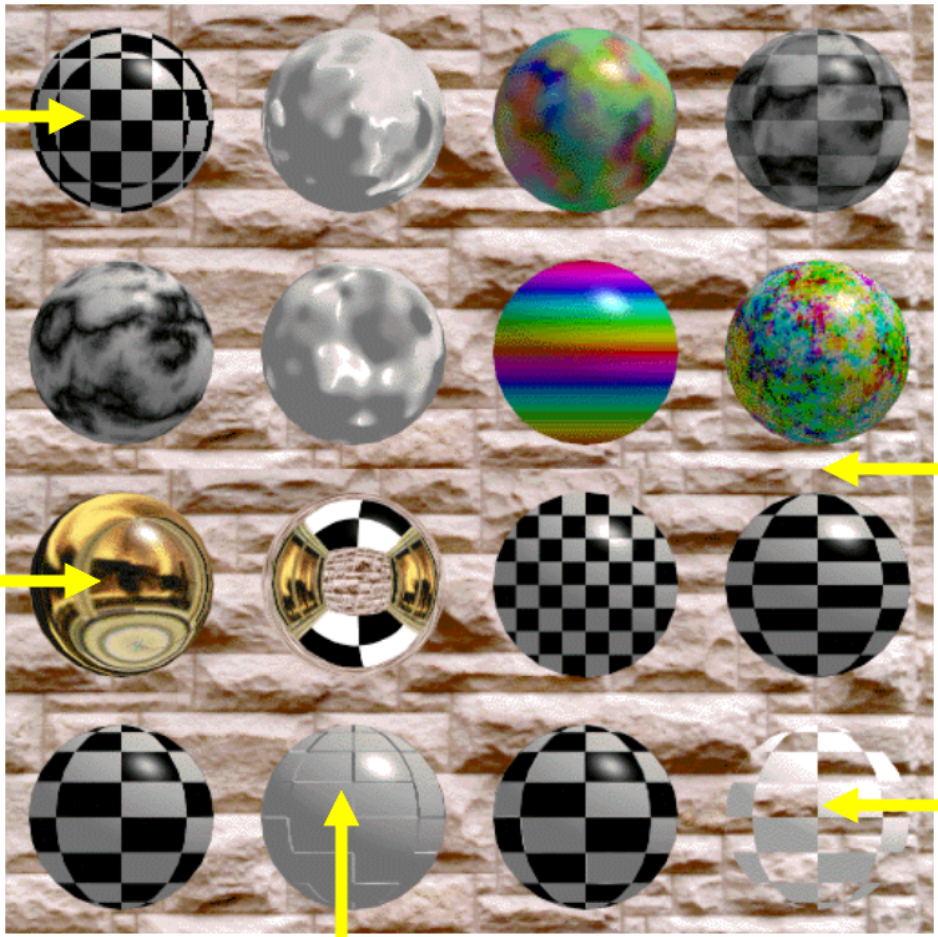
Multi-texturing:

- ✓ Several texels can be overlaid on one pixel;
- ✓ A texture ***blending cascade*** is made up of a series of texture stages



(from "Real Time Rendering")

Allow more complex textures



Normal texture

Background texture

Reflectivity texture

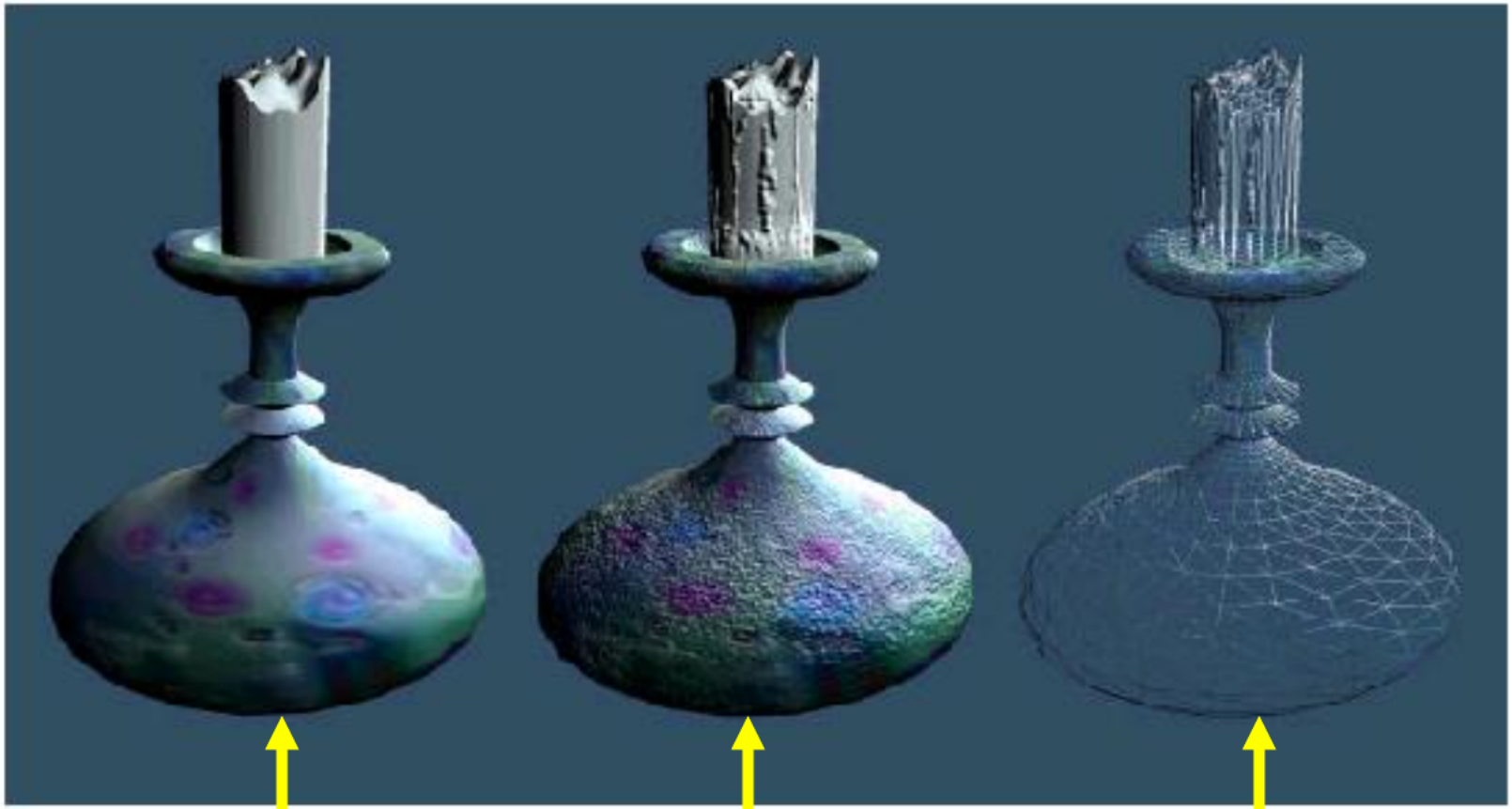
Transparency texture

Bump maps

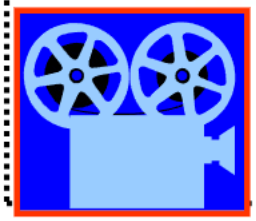
Multi-texturing for bump mapping:

- ✓ Lighting effects caused by irregularities on object surface are simulated through "***bump mapping***";
- ✓ This encodes surface irregularities as textures;
- ✓ *No change in model geometry*. No added computations at the geometry stage;
- ✓ Done as part of the per-pixel shading operations of the Nvidia Shading Rasterizer

Bump mapping per-pixel shading



VC 5.1



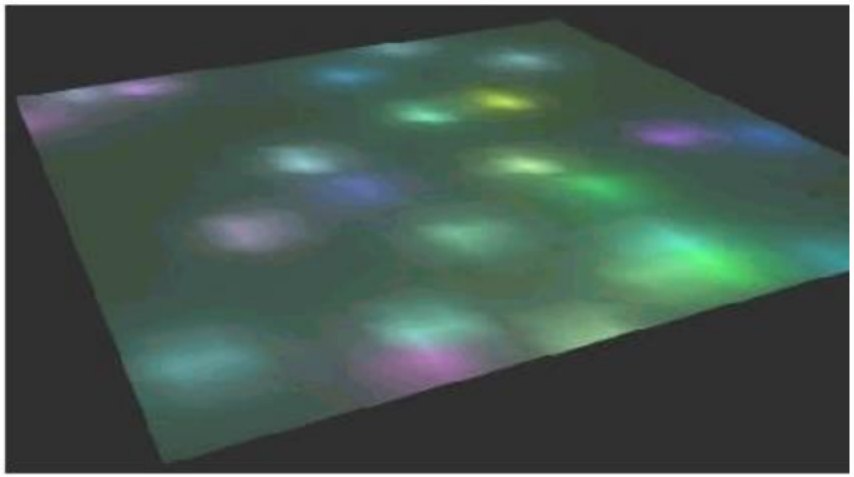
Normal texture

Multi-texture

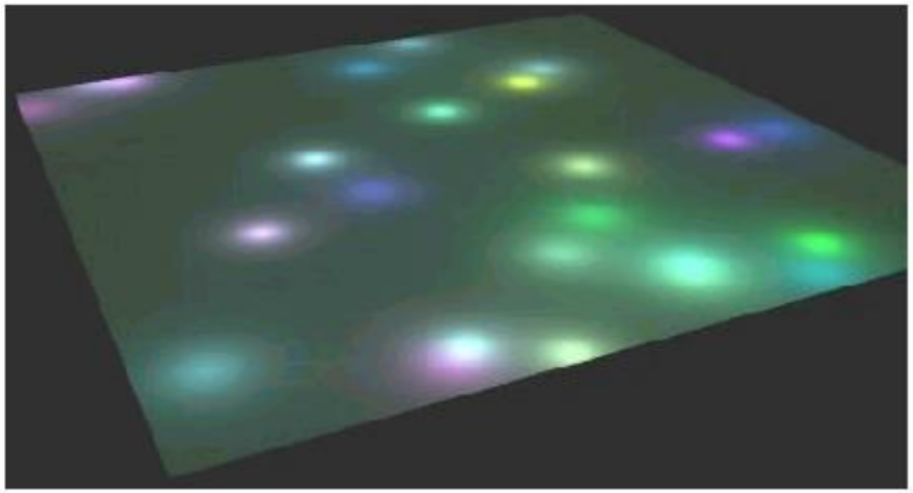
Bump texture

Multi-texturing for lighting:

- ✓ Several texels can be overlaid on one pixel;
- ✓ One application in more realistic lighting;
- ✓ Polygonal lighting is real-time but requires lots of polygons (triangles) for realistic appearance



Vertex lighting of low polygon count surface – lights are diffuse – tessellated.

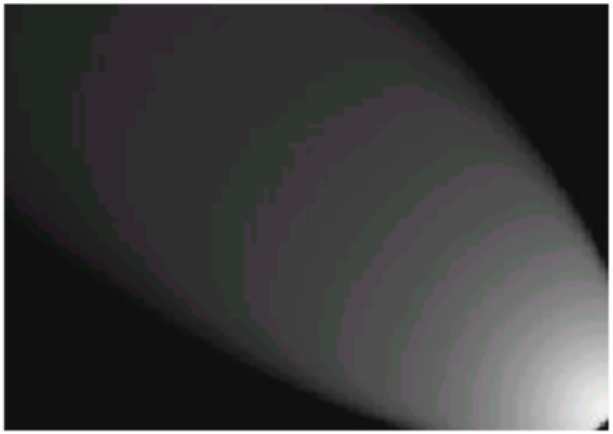


Vertex lighting of high polygon count surface – lights have realistic appearance. High computation load

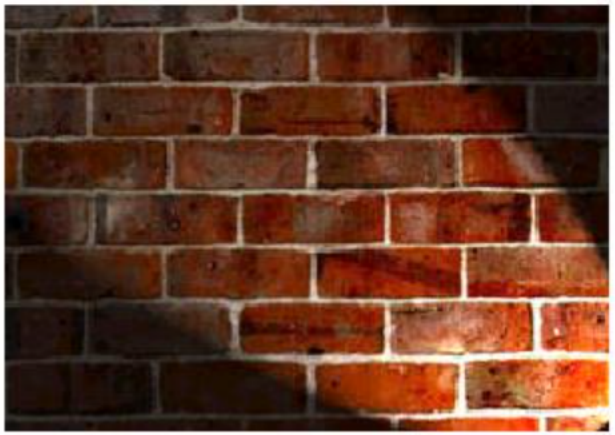
(from NVIDIA technical brief)

Multi-texturing (texture blending):

- ✓ Realistic-looking lighting can be done with 2-D textures called “light maps”;
- ✓ Not applicable to real-time (need to be recomputed when object moves)



Standard lighting map 2-D texture



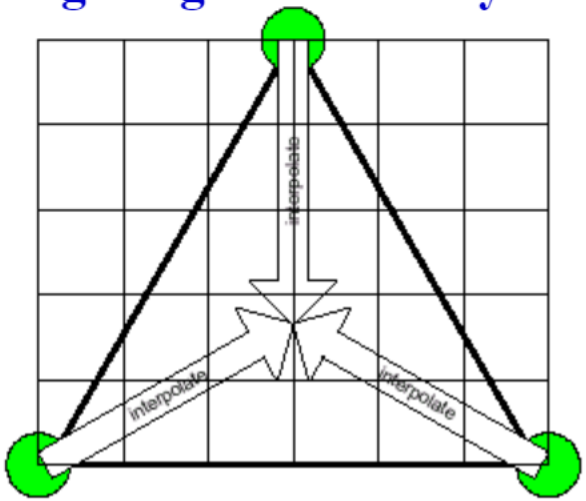
Light map texture overlaid on top of wall texture. Realistic and low polygon count. Not real-time!

(from NVIDIA technical brief)

NVIDIA Shading Rasterizer:

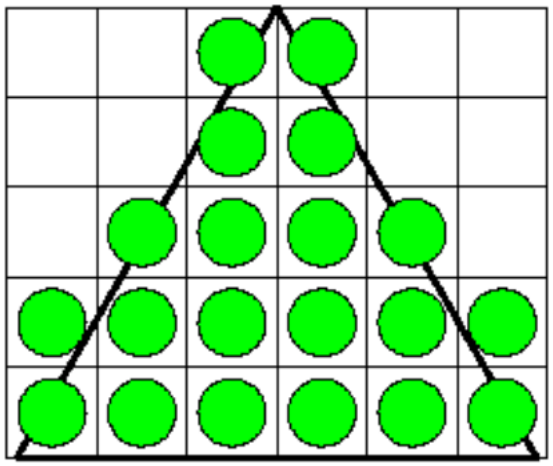
- ✓ NVIDIA proposed to combine the dynamics of Polygon-based shading with the realism of light maps;
- ✓ Vertex info used in shading: position, texture, normal;
- ✓ Let's assign same info to pixels and do shading at ***pixel level***!

Lighting in Geometry Stage



Lighting by interpolation of vertex information

Lighting with Shading Rasterizer



Realistic and real-time!

(from NVIDIA technical brief)

NVIDIA Shading Rasterizer (NSR):

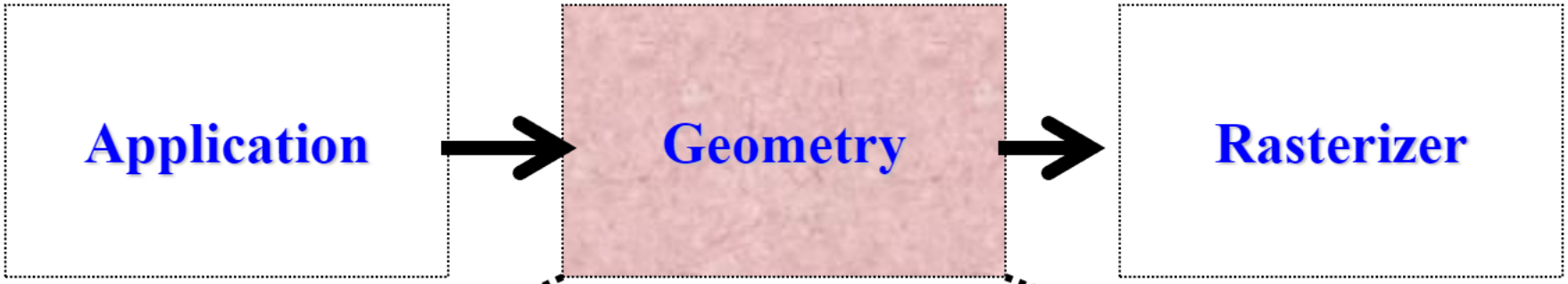
- ✓ Texels contain information about position and color;
- ✓ We still need information about normals – special type of texture called "***normal map***"
- ✓ texel information + normal maps are used to calculate light intensity and direction on a per-pixel basis;
- ✓ Texture blending operations allow real-time diffuse, spot and point light effects for each pixel;
- ✓ Much faster. NVIDIA Quadro2 can render in real time frames that took hours of off-line rendering for Pixar's ***A Bug's Life***.

(from NVIDIA technical brief)

VR Kinematics Modeling:

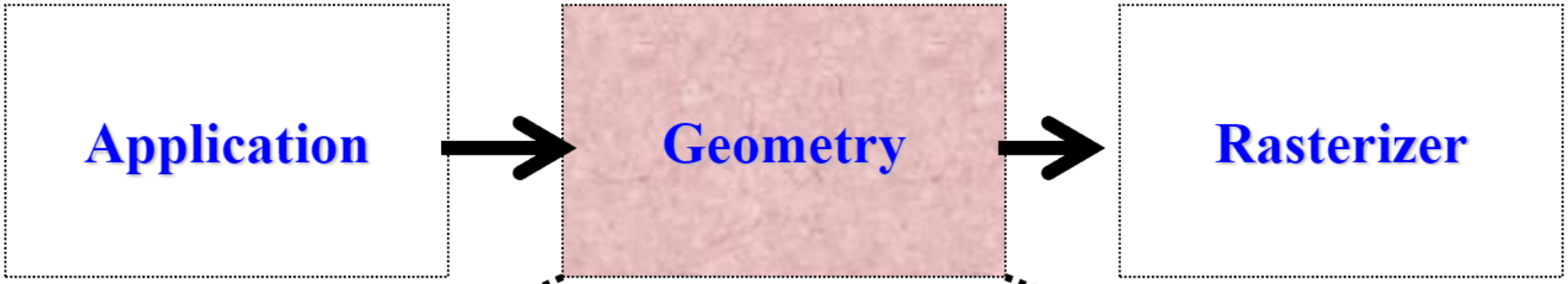
- ✓ Homogeneous transformation matrices;
- ✓ Object position;
- ✓ Transformation invariants;
- ✓ Object hierarchies;
- ✓ **Viewing the 3-D world.**

The Graphics Rendering Pipeline (revisited)



The Geometry Functional Sub-Stages

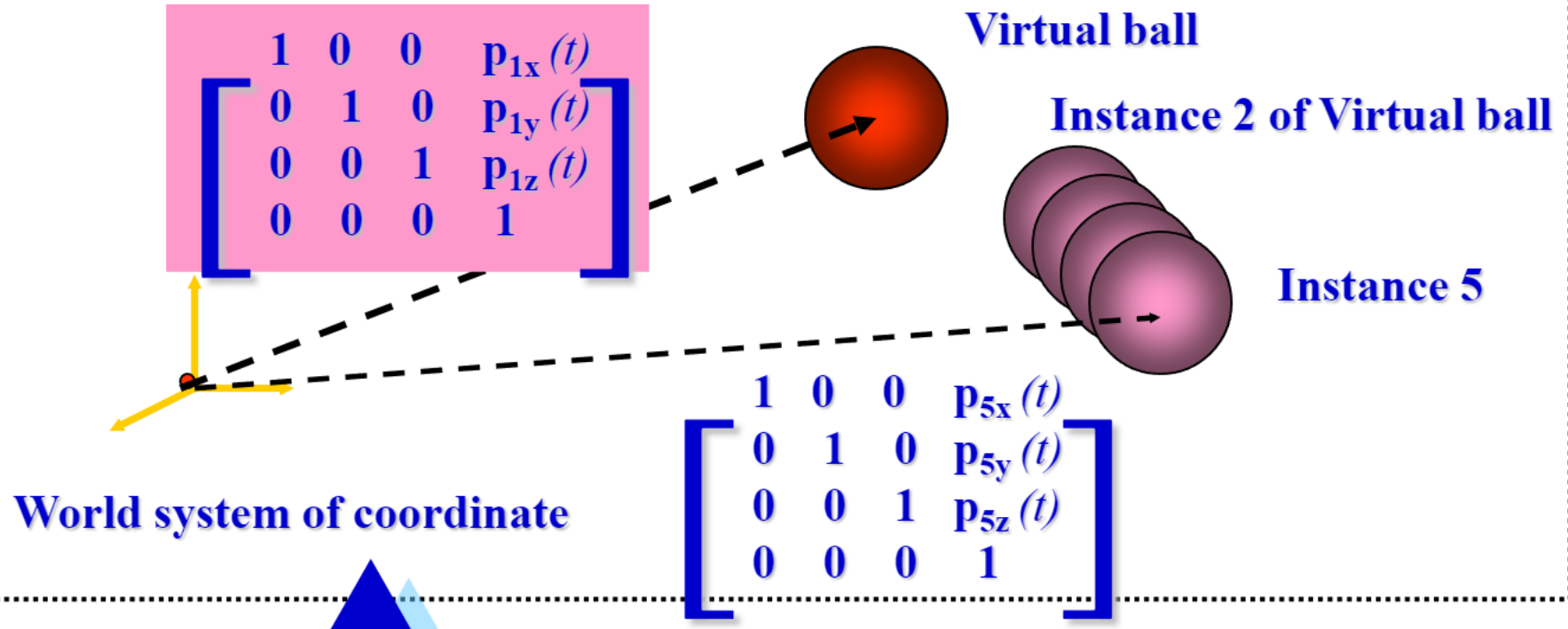
The Rendering Pipeline



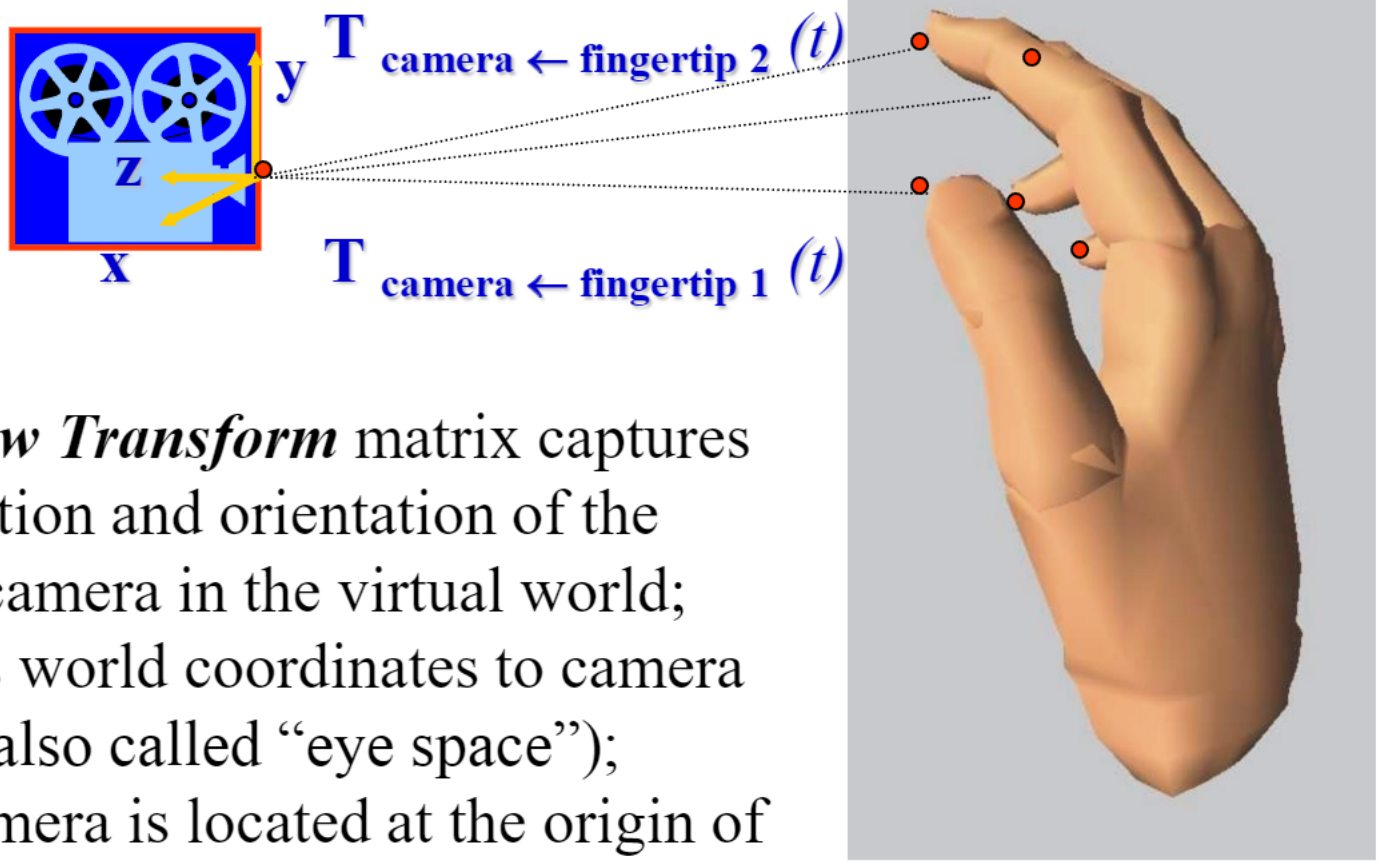
The Geometry Functional Sub-Stages

Model and Viewing Transformations:

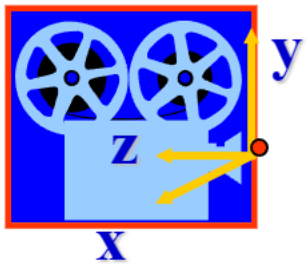
- ✓ Model transforms link object coordinates to world coordinates. By changing the model transform, the same object can appear several times in the scene.
  We call these ***instances***.



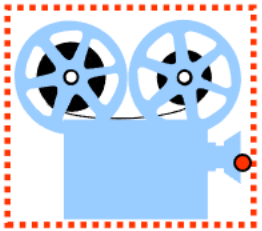
Camera system of coordinates



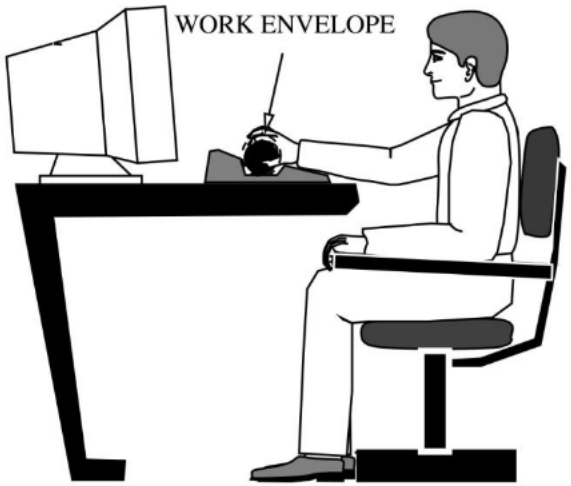
- The ***View Transform*** matrix captures the position and orientation of the virtual camera in the virtual world;
- It maps world coordinates to camera space (also called "eye space");
- The camera is located at the origin of the camera coordinate system, looking in the negative ***Z*** axis, with ***Y*** pointing upwards, and ***X*** to the right.



Camera "fly-by"

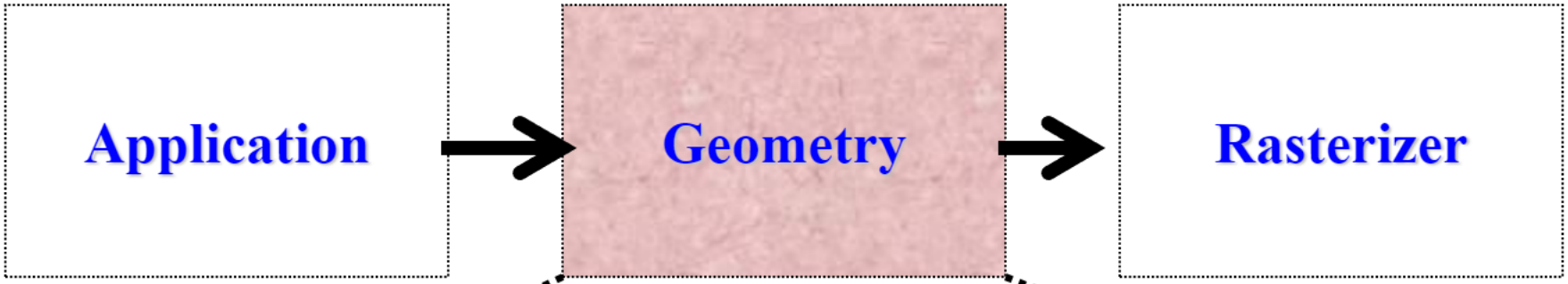


$\mathbf{T}_{\text{camera} \leftarrow \text{tree}}(t)$



User interactively changes the ***viewing transform*** by changing the position and aim of the virtual camera = "fly-by." Example – NVIDIA Grove demo. This demo Also shows ***instances***

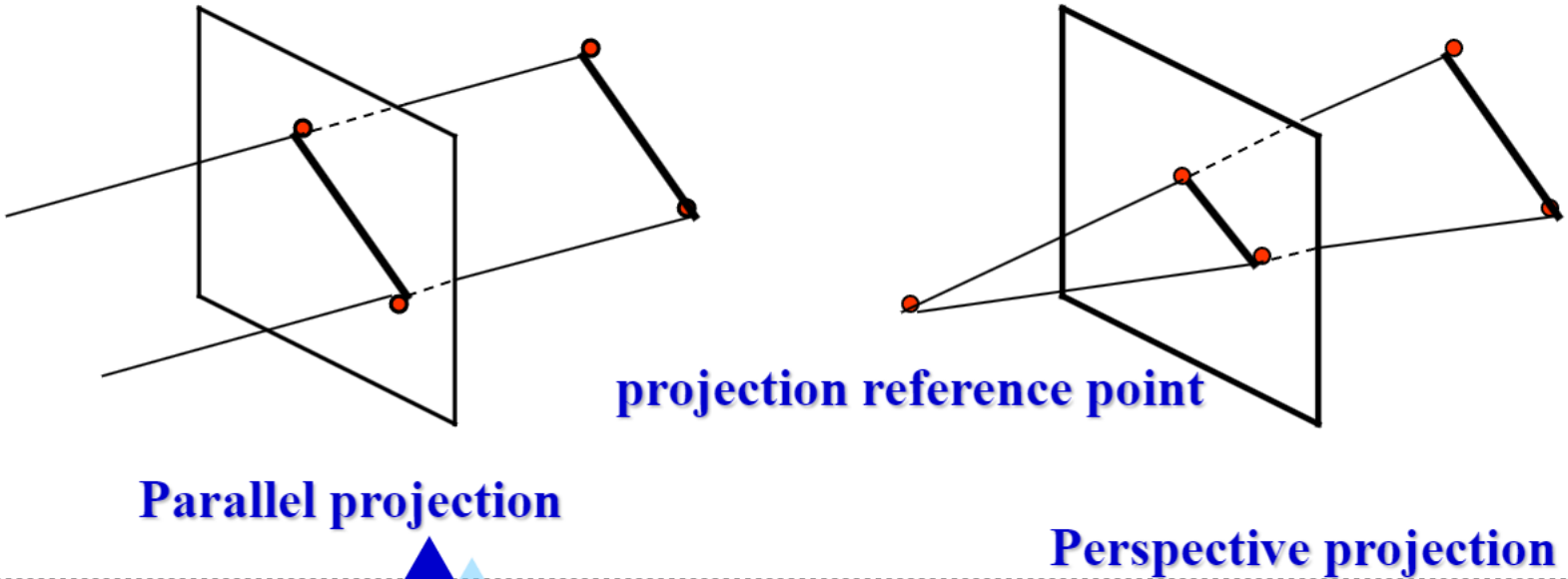
The Rendering Pipeline



The Geometry Functional Sub-Stages

Projection Transformations:

✓ Models what portion (volume) of the virtual world the camera actually sees. There are two kinds of projections, ***parallel*** projection and ***perspective*** projection. VR uses perspective.

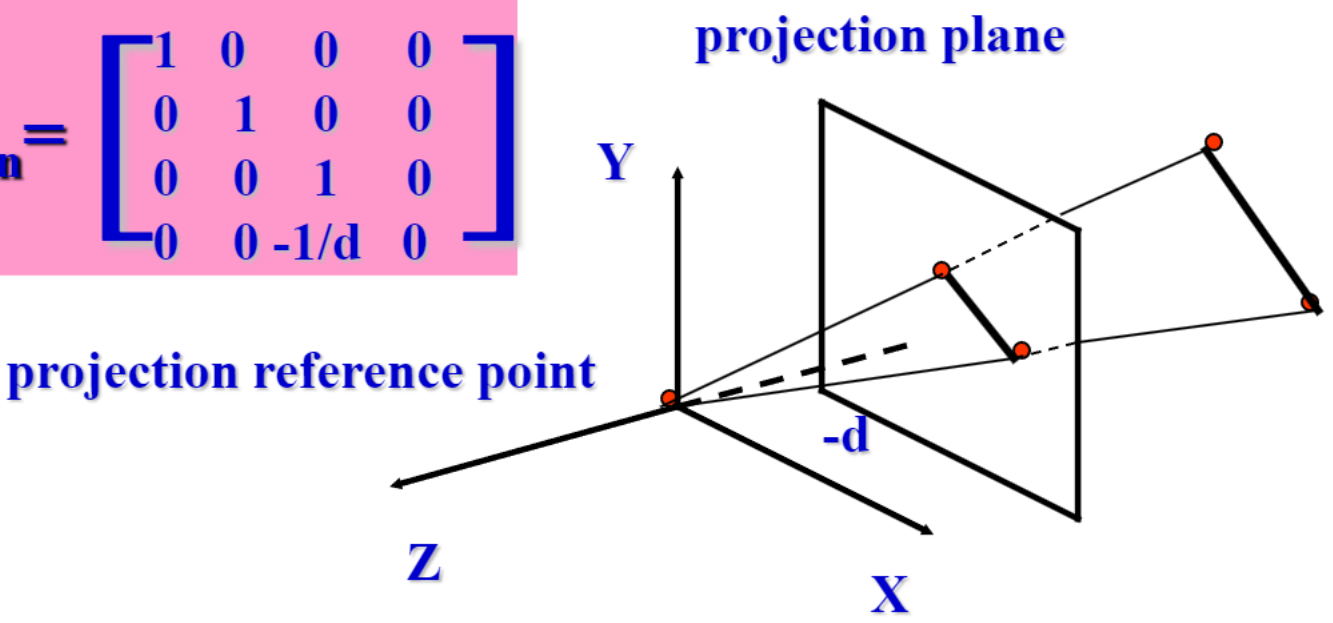


Perspective Projection Transformation:

✓ If the projection reference point is at the origin of the system of coordinates, and the projection plane is at ***–d,***

Then the (non invertible) perspective projection transformation matrix is:

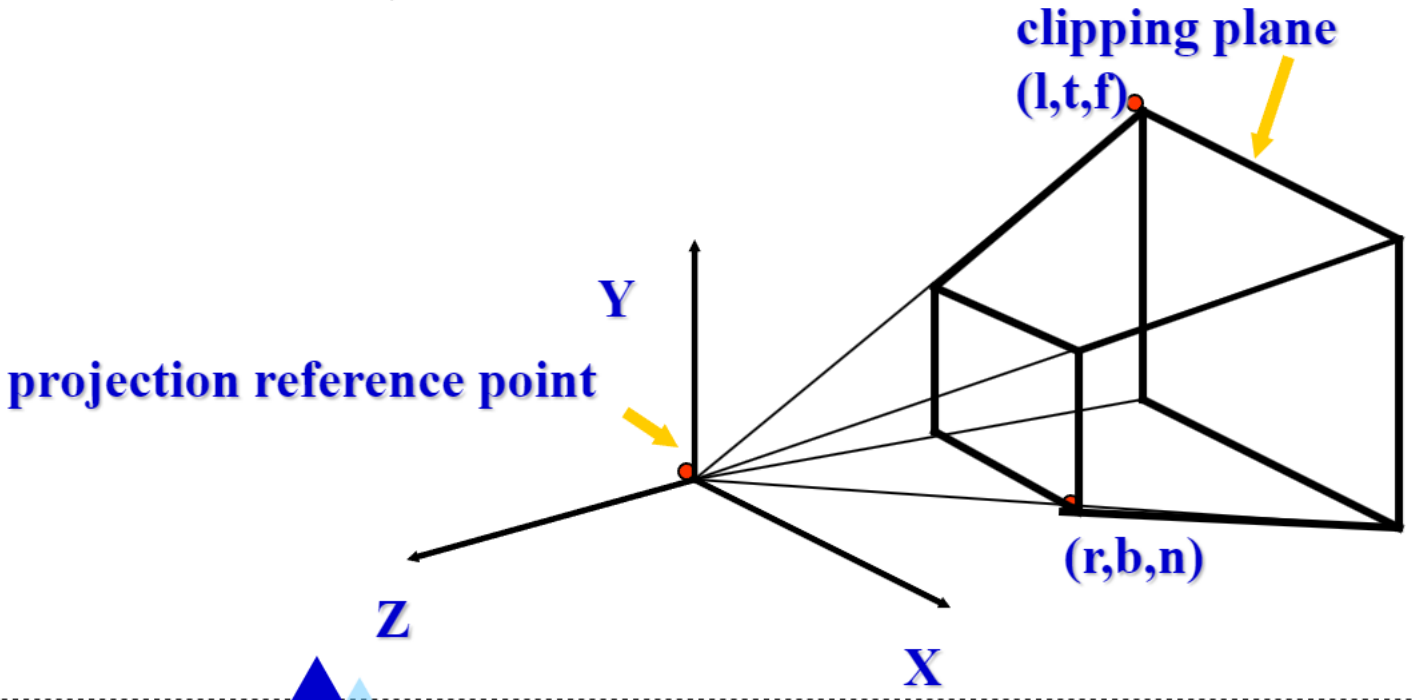
$$T_{projection} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -1/d & 0 \end{bmatrix}$$



Second Perspective Projection Transformation:

✓ The portion of the virtual world seen by the camera at a given time is limited by front and back "*clipping planes*".

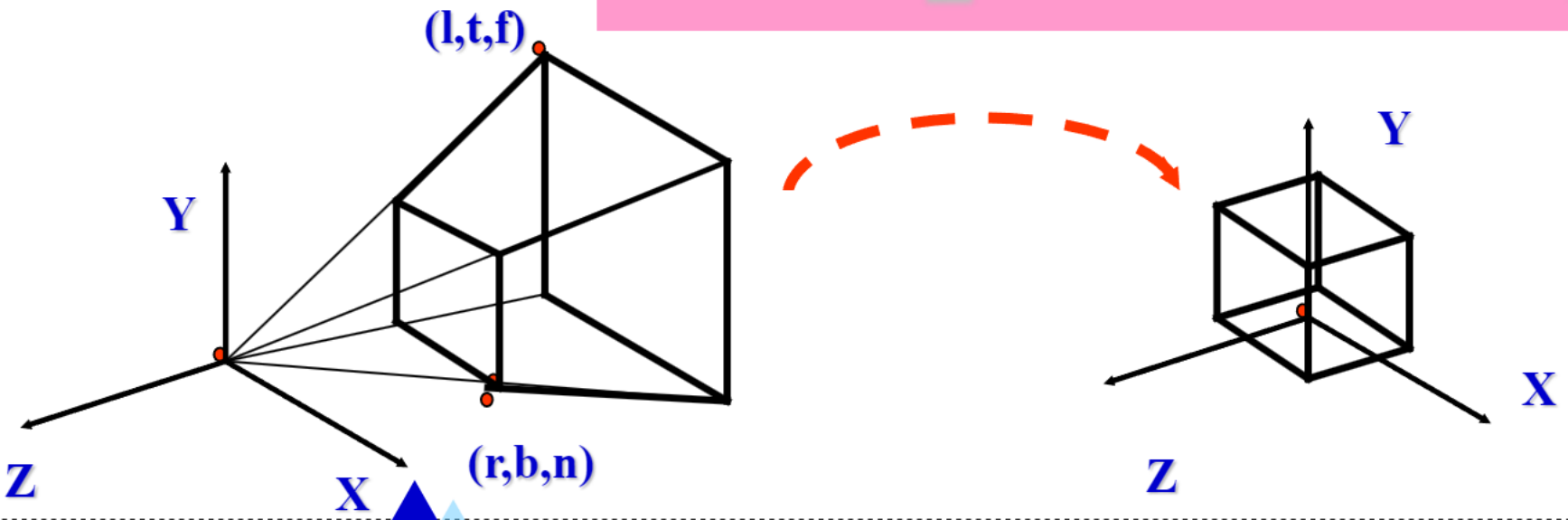
These are at ***z=n*** and ***z= f***. Only what is within the viewing cone (also called *fulcrum*) is sent down the rendering pipe.



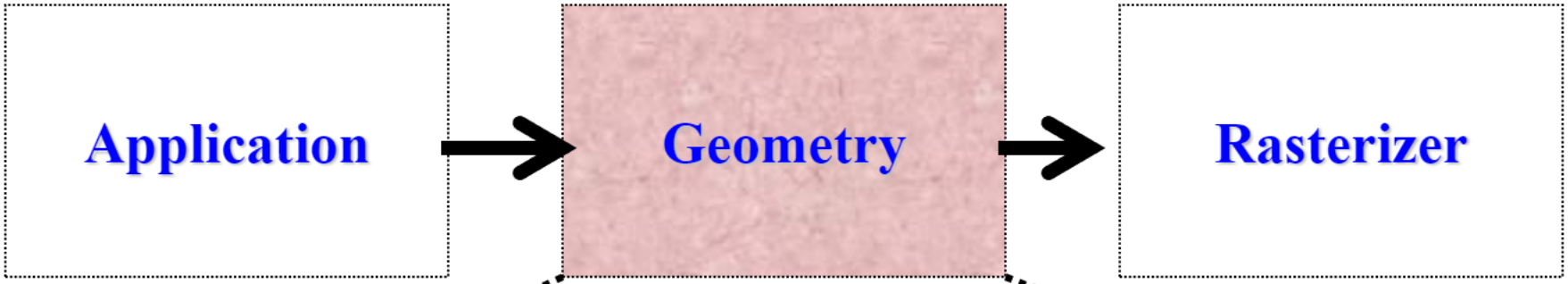
Canonical Mapping:

✓ The second projection transform maps the viewing volume to a unit cube with extreme points at (-1,-1,-1) and (1,1,1). This is called the ***canonical view volume***.

$$T'_{projection} = \begin{bmatrix} \frac{2n}{r-l} & 0 & -\frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & -\frac{t+b}{t-b} & 0 \\ 0 & 0 & \frac{f+n}{f-n} & -\frac{2fn}{f-n} \\ 0 & 0 & 1 & 0 \end{bmatrix}$$



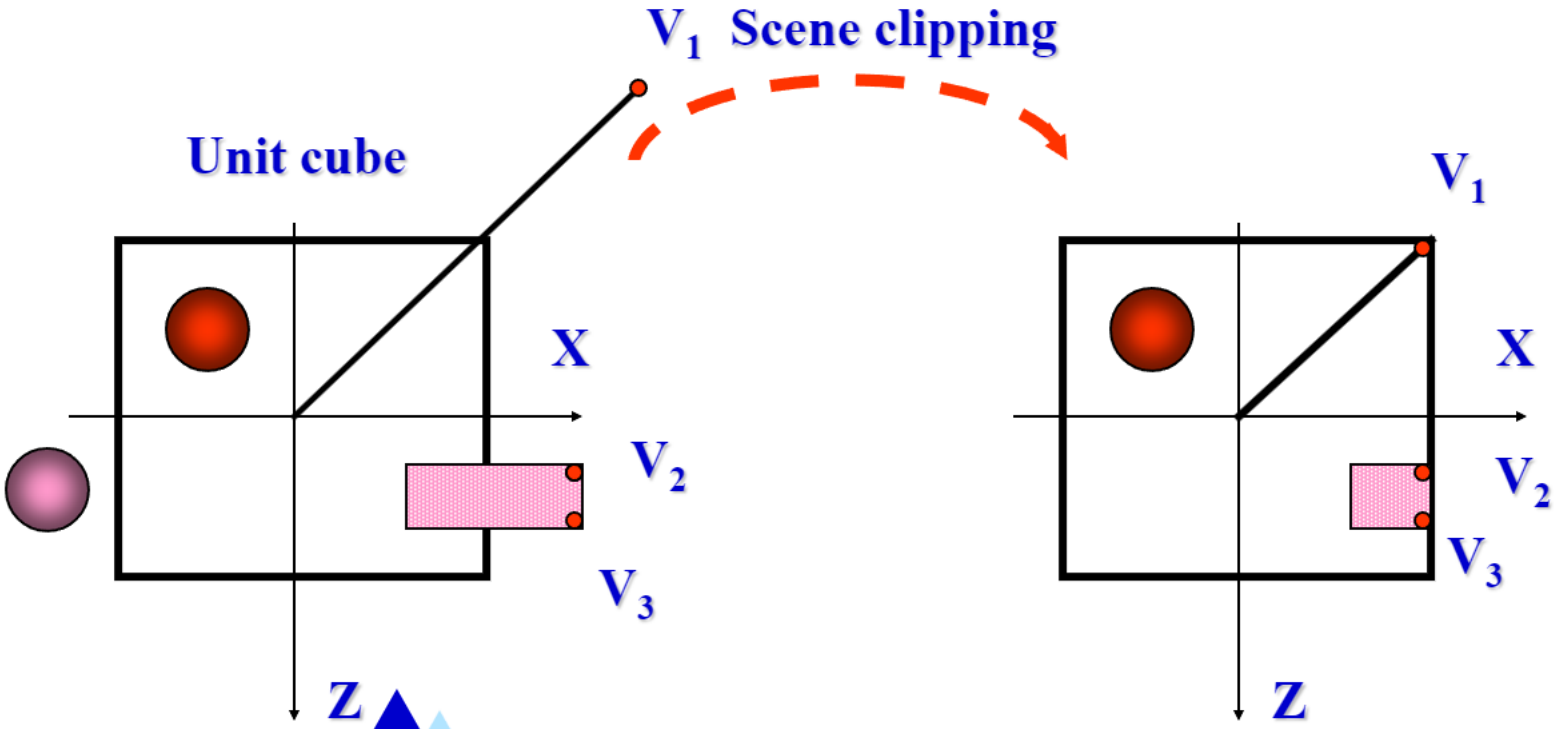
The Rendering Pipeline



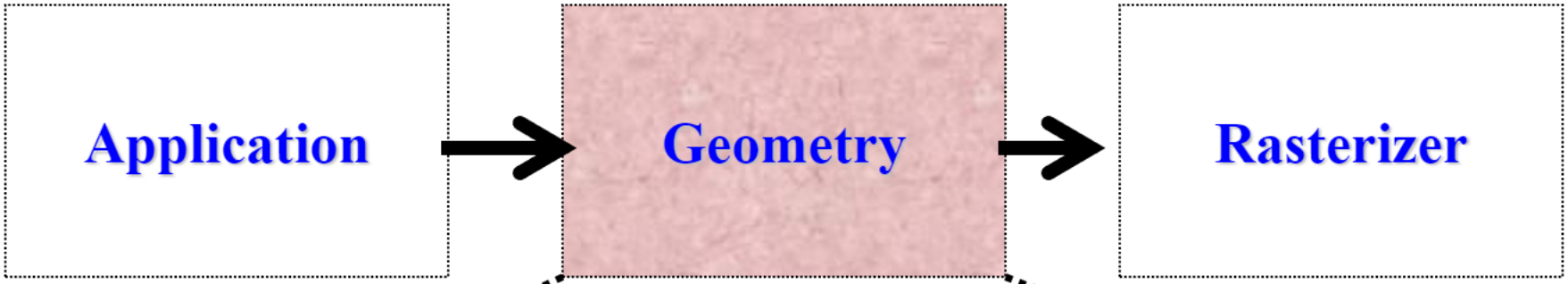
The Geometry Functional Sub-Stages

Clipping Transformation:

✓ Since the fulcrum maps to the unit cube, only objects inside it will be rendered. Some objects are *partly* inside the unit cube (ex. the line and the rectangle). Then they need to be "clipped". The vertex V_1 is replaced by new one at the intersection between the line and the viewing cone, etc.



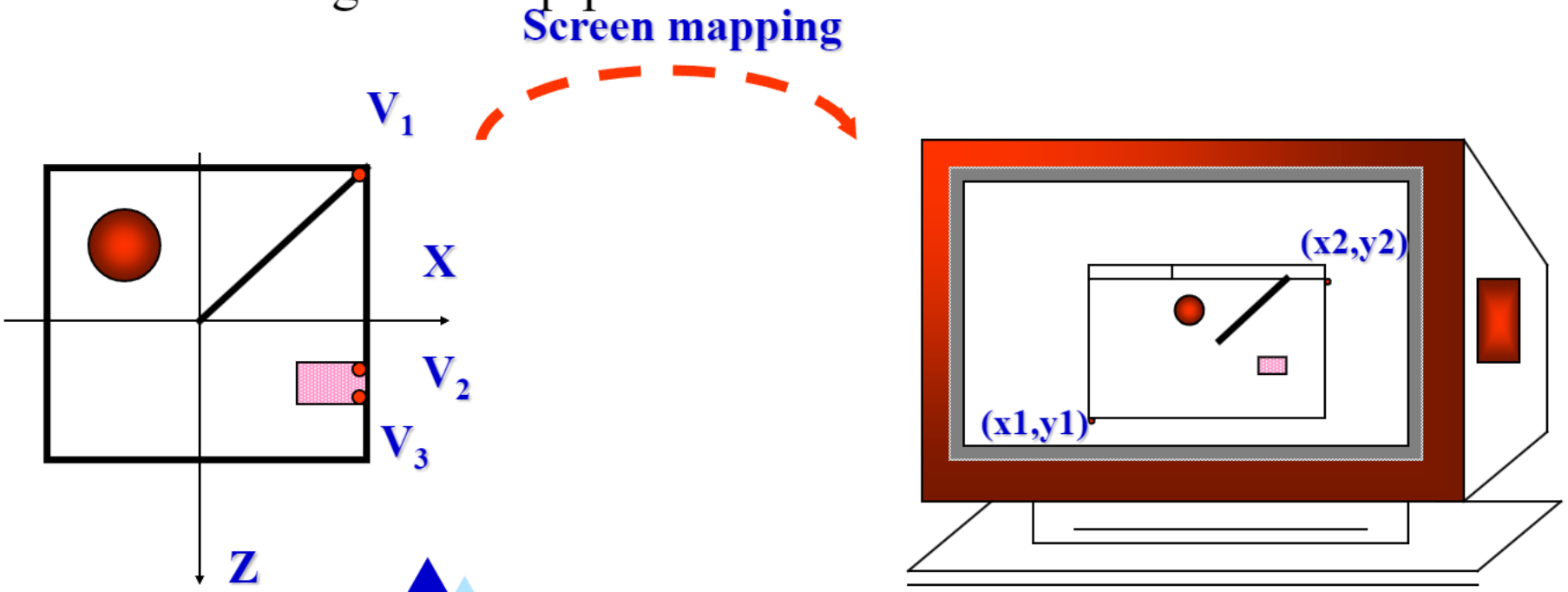
The Rendering Pipeline



The Geometry Functional Sub-Stages

Screen Mapping (Viewport Transformation):

- ✓ The scene is rendered into a window with corners (x_1,y_1), (x_2,y_2)
- ✓ Screen mapping is a translation followed by a scaling that affects the **x** and **y** coordinates of the primitives (objects), but not their z coordinates. Screen coordinates plus $z \in [-1,1]$ are passed to the rasterizer stage of the pipeline.



KINEMATICS MODELING:

- ✓ Homogeneous transformation matrices;
- ✓ Object position;
- ✓ Transformation invariants;
- ✓Object hierarchies;
- ✓ Viewing the 3-D world.

Homogeneous Transformations:

- ✓ Homogeneous system of coordinates is a right-hand Cartesian system of coordinates with ***orthonormal*** unit vector triads;
- ✓ Such **(i, j, k)** triads have the property that their norms $|\mathbf{i}| = |\mathbf{j}| = |\mathbf{k}| = 1$ and their dot product is $\mathbf{i} \bullet \mathbf{j} = \mathbf{i} \bullet \mathbf{k} = \mathbf{j} \bullet \mathbf{k} = 0$;
- ✓ Homogeneous transformation matrices relate two such systems through a 4×4 matrix.

Homogeneous Transformations:

- ✓ Have the general format:

$$T_{A \leftarrow B} = \begin{bmatrix} & R_{3x3} & & P_{3x1} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $\mathbf{R_{3x3}}$ is the ***rotation submatrix*** expressing the orientation of the system of coordinates **B** vs. system of coordinates **A;**

$\mathbf{P_{3x1}}$ is the ***position vector*** of the origin of system **B** vs. the origin of system of coordinates **A**.

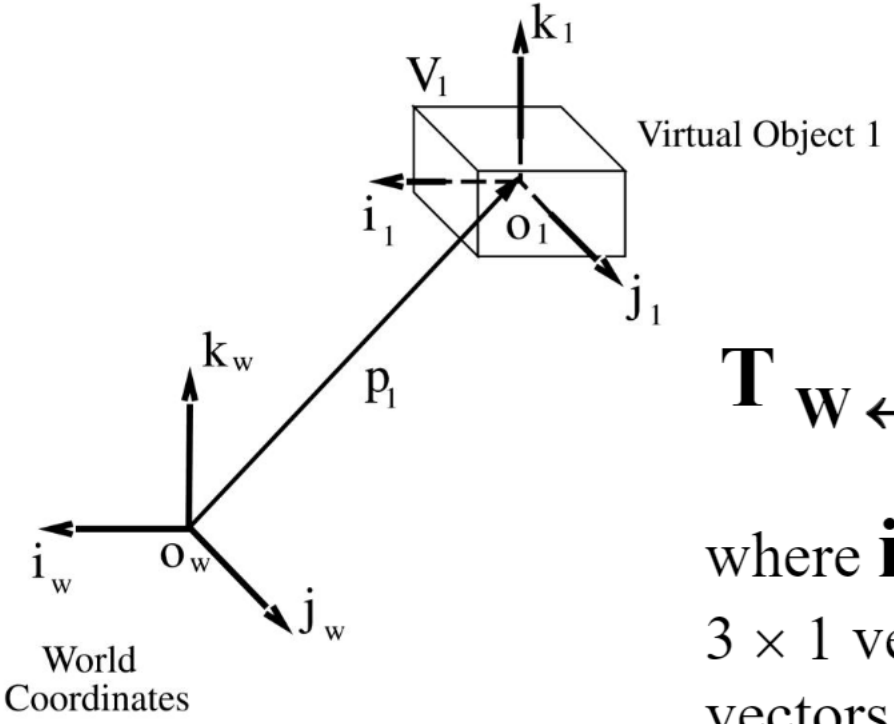
Homogeneous Transformations:

- ✓ Have many advantages:
- treat object translation and rotation mathematically in the same way;
- are easily invertible;

$$T_{B \leftarrow A} = (T_{A \leftarrow B})^{-1} = \begin{bmatrix} & R^T & & -R^T \bullet P \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Object Position/Orientation (static):

- ✓ given by homogeneous transformation matrix that relates the object system of coordinates to world system of coordinates.



$$\mathbf{T}_{\mathbf{w}\leftarrow\mathbf{1}} = \begin{bmatrix} \mathbf{i}_{\mathbf{w}\leftarrow\mathbf{1}} & \mathbf{j}_{\mathbf{w}\leftarrow\mathbf{1}} & \mathbf{k}_{\mathbf{w}\leftarrow\mathbf{1}} & \mathbf{P}_{\mathbf{1}} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

where $\mathbf{i}_{\mathbf{w}\leftarrow\mathbf{1}}$, $\mathbf{j}_{\mathbf{w}\leftarrow\mathbf{1}}$, $\mathbf{k}_{\mathbf{w}\leftarrow\mathbf{1}}$ are 3×1 vectors projecting the object unit vectors into world system of coordinates

Object Position/Orientation (moving):

✓ If the virtual object moves, then the transformation matrix becomes a *function of time*

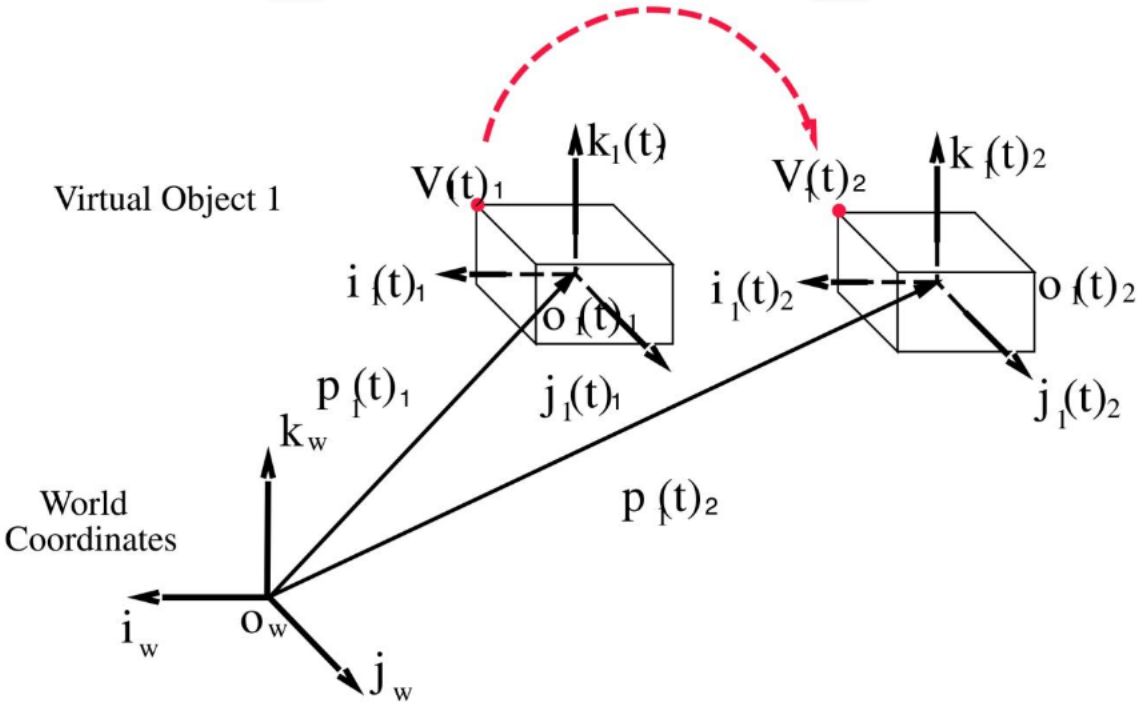
$$\mathbf{T}_{\mathbf{W}\leftarrow\mathbf{1}}(t) = \begin{bmatrix} \mathbf{i}_{\mathbf{w}\leftarrow\mathbf{1}}(t) & \mathbf{j}_{\mathbf{w}\leftarrow\mathbf{1}}(t) & \mathbf{k}_{\mathbf{w}\leftarrow\mathbf{1}}(t) & \mathbf{P}_1(t) \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix}$$

✓ The position of an object vertex $\mathbf{V}_i$ in world coordinates versus its position in object coordinates

$$\mathbf{V}_i^{(\mathbf{W})}(t) = \mathbf{T}_{\mathbf{W}\leftarrow\mathbf{1}}(t)\,\mathbf{V}_i^{(\mathbf{object})}$$

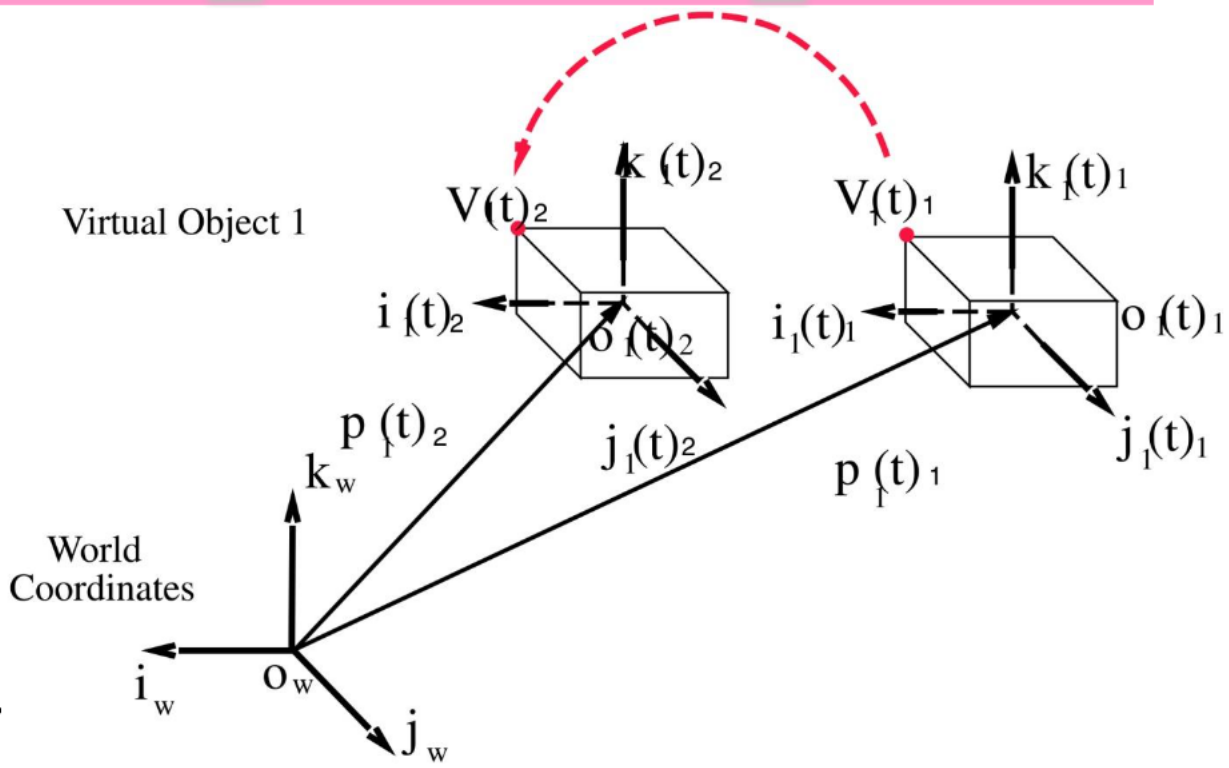
If the aligned virtual object translates, all vertices translate

$$\mathbf{V}_i^{(W)}(t) = \begin{bmatrix} 1 & 0 & 0 & p_{1x}(t) \\ 0 & 1 & 0 & p_{1y}(t) \\ 0 & 0 & 1 & p_{1z}(t) \\ 0 & 0 & 0 & 1 \end{bmatrix} \mathbf{V}_i^{(object)}$$



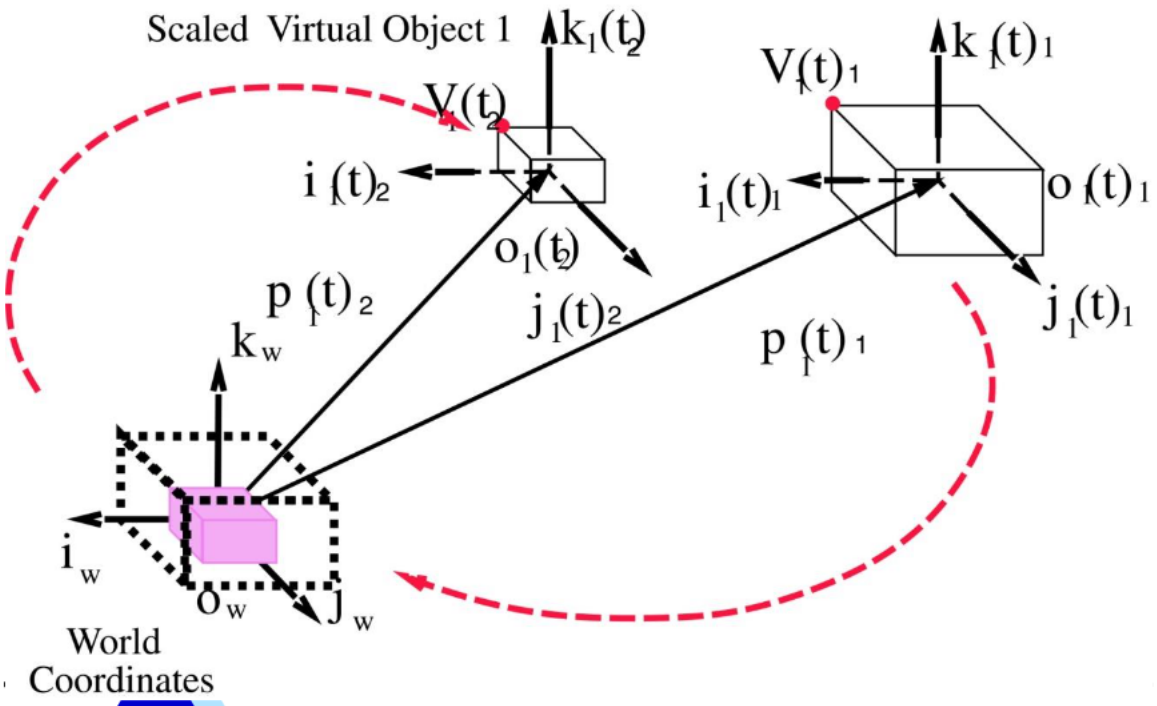
If the virtual object translates *back* to its initial position, all its vertices translate by an equal but negative amount.

$$V_i^{(W)}(t) = \begin{bmatrix} 1 & 0 & 0 & -p_{1x}(t) \\ 0 & 1 & 0 & -p_{1y}(t) \\ 0 & 0 & 1 & -p_{1z}(t) \\ 0 & 0 & 0 & 1 \end{bmatrix} V_i^{(object)}$$

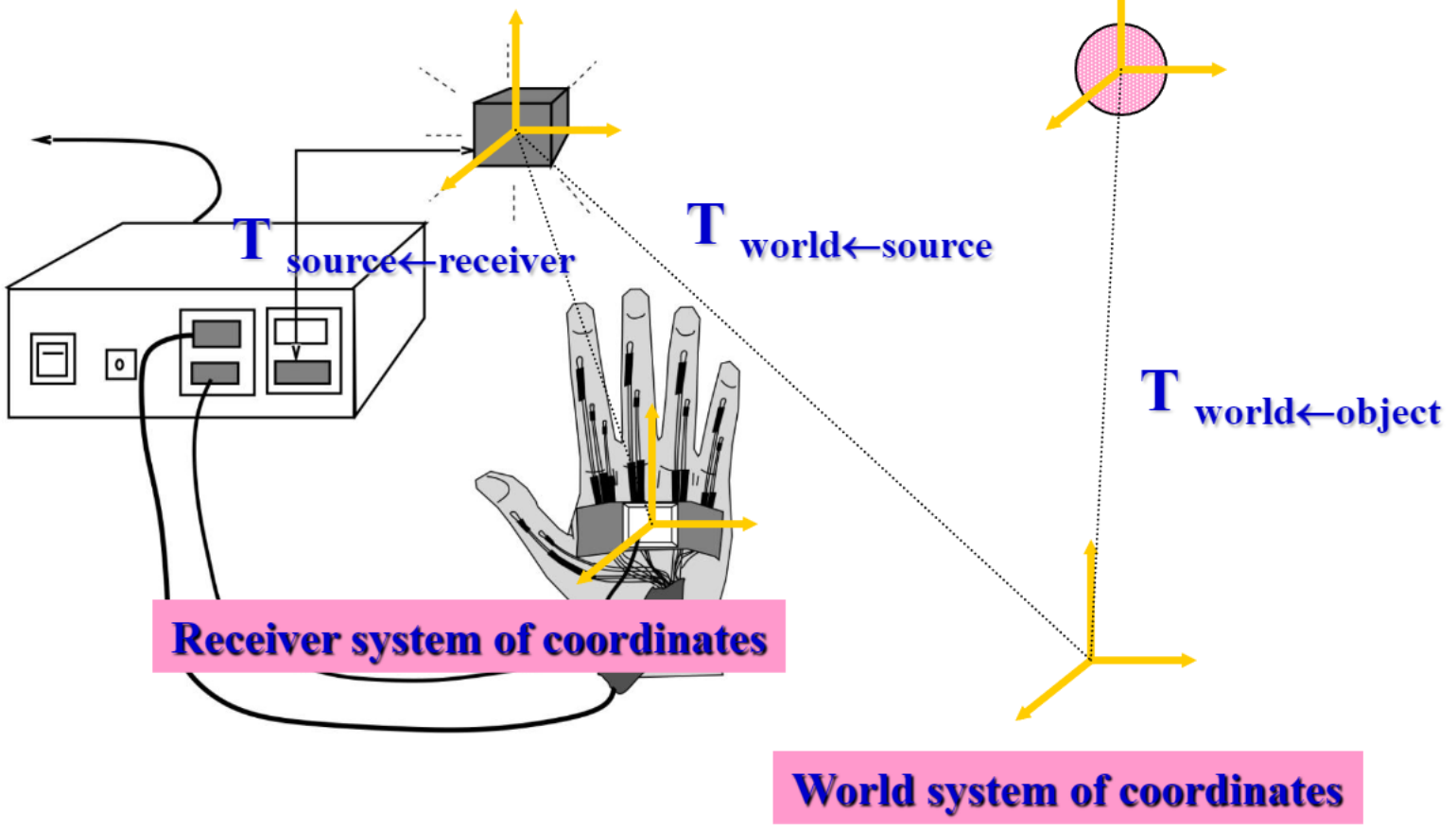


If the virtual object needs to be scaled, it is translated to origin, scaled, then translated to back

$$V_i^{(W)}(t) = T_{w \leftarrow 2} \begin{bmatrix} s_i & 0 & 0 & 0 \\ 0 & s_j & 0 & 0 \\ 0 & 0 & s_k & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} T_{w \leftarrow 1} V_i^{(object)}$$



Tracking a virtual hand:



Transformations concatenation:

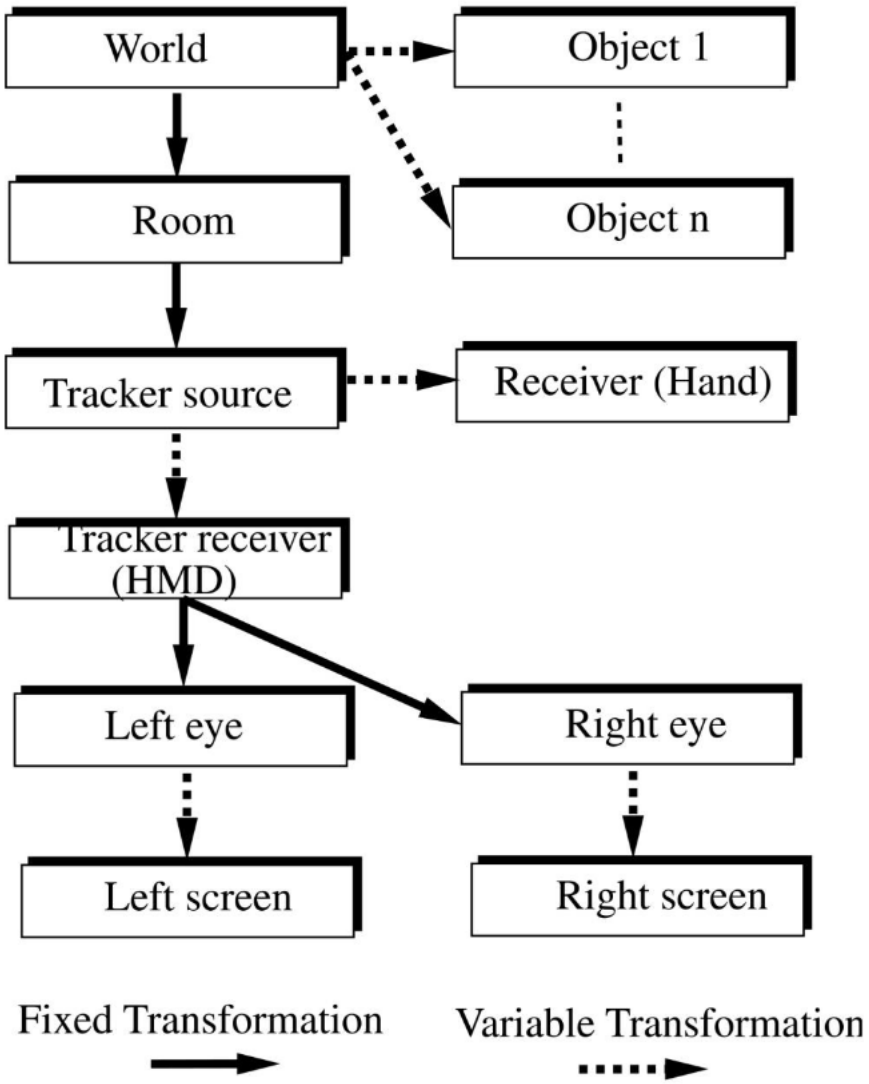
✓ Transformation matrices can be compounded to obtain the resulting motion. Example – simulating a virtual hand.

$$T_{W \leftarrow hand}(t) = T_{W \leftarrow source}\, T_{source \leftarrow receiver}(t)$$

✓ If the object is grasped, then its position does not change vs. the hand. Thus the movement of the grasped object in word coordinates is:

$$T_{W \leftarrow object}(t) = T_{W \leftarrow source}\, T_{source \leftarrow receiver}(t)\, T_{receiver \leftarrow object}$$

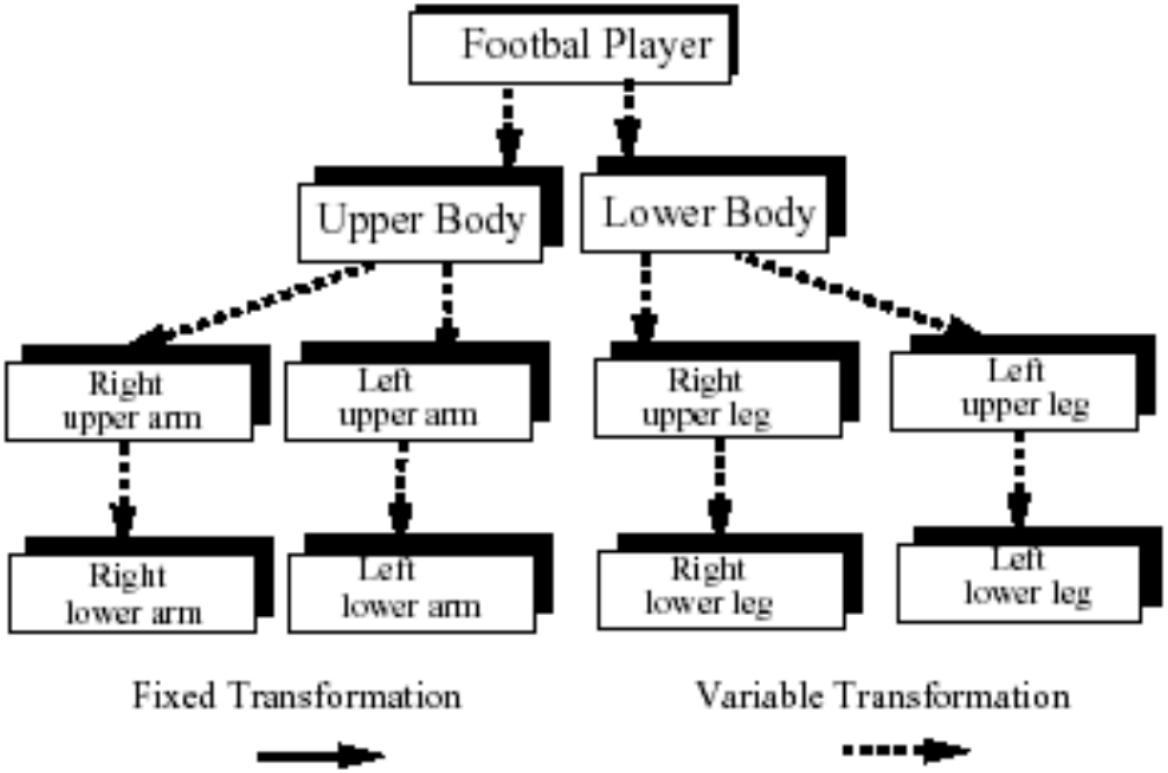
VR Kinematics Modeling



Object Hierarchies:

- ✓ Allows models to be partitioned into a hierarchy, and become dynamic;
- ✓ Segments are either parent (higher level object) or child (lower level objects).
- ✓ The motion of a parent is replicated by its children but not the other way around.
- ✓ Example – the virtual human and the virtual hand;
- ✓ At the top of the hierarchy is the "world global transformation" that determines the view to the scene.

VR Kinematics Modeling



a) b)

Model hierarchy: a) static model (Viewpoint Datalabs); b) segmented model.

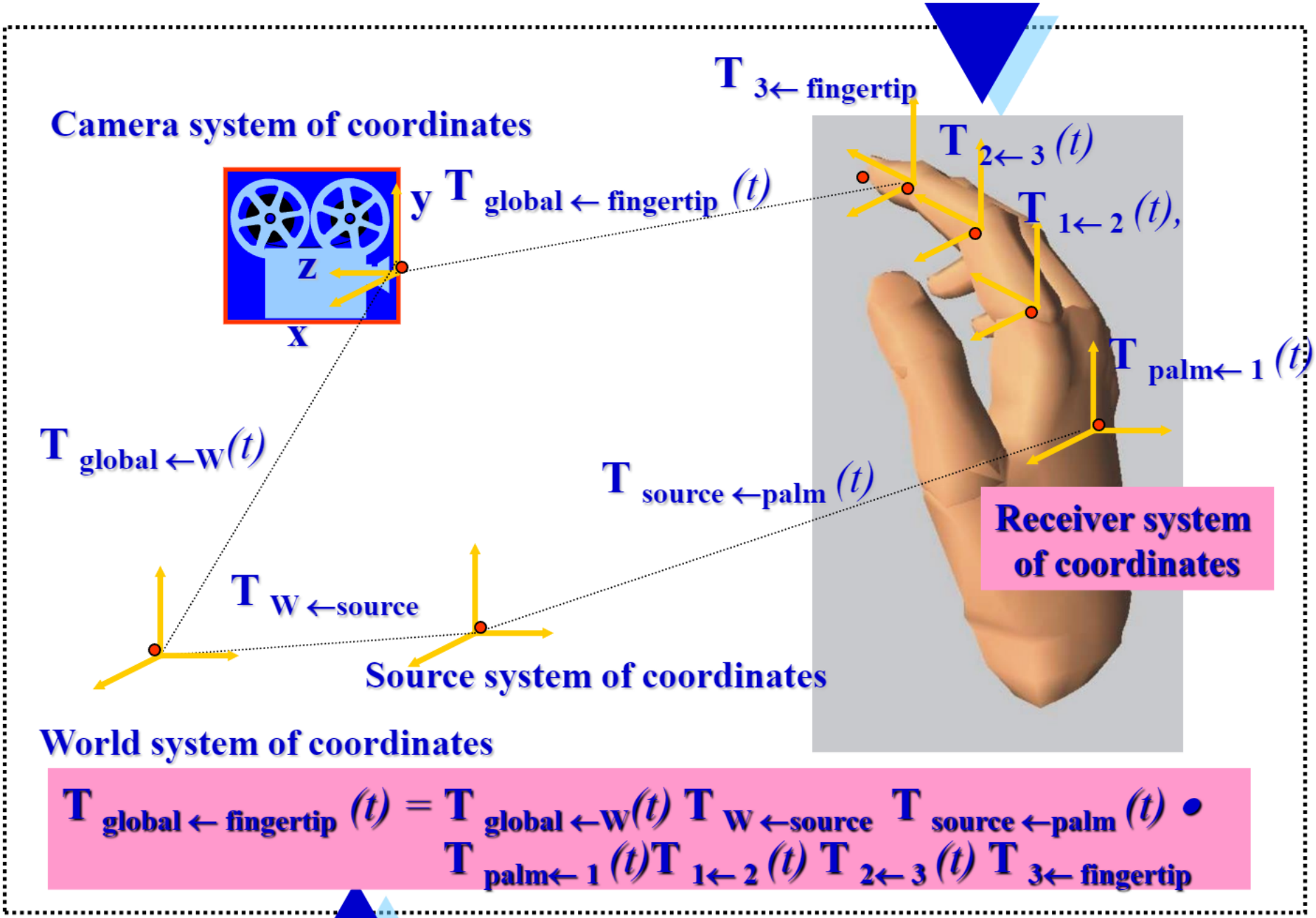
Object hierarchy for a Virtual Hand:

✓ Transformation matrices can be compounded to obtain the motion of the fingertip versus the world coordinates.

$$T_{global \leftarrow fingertip}(t) = T_{global \leftarrow W}(t)\, T_{W \leftarrow source}\, T_{source \leftarrow palm}(t) \bullet T_{palm \leftarrow 1}(t) T_{1 \leftarrow 2}(t)\, T_{2 \leftarrow 3}(t)\, T_{3 \leftarrow fingertip}$$

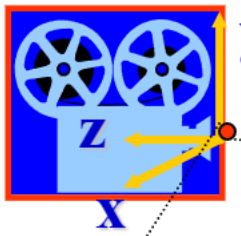
$T_{W \leftarrow palm}(t)$ is given by the glove tracker

$T_{palm \leftarrow 1}(t)$, $T_{1 \leftarrow 2}(t)$, $T_{2 \leftarrow 3}(t)$ are given by the sensors on the glove



$$T_{global \leftarrow fingertip}(t) = T_{global \leftarrow W}(t)\, T_{W \leftarrow source}\, T_{source \leftarrow palm}(t) \bullet T_{palm \leftarrow 1}(t) T_{1 \leftarrow 2}(t)\, T_{2 \leftarrow 3}(t)\, T_{3 \leftarrow fingertip}$$

Camera system of coordinates



$T_{global \leftarrow car(t)}$

$T_{global \leftarrow W(t)}$

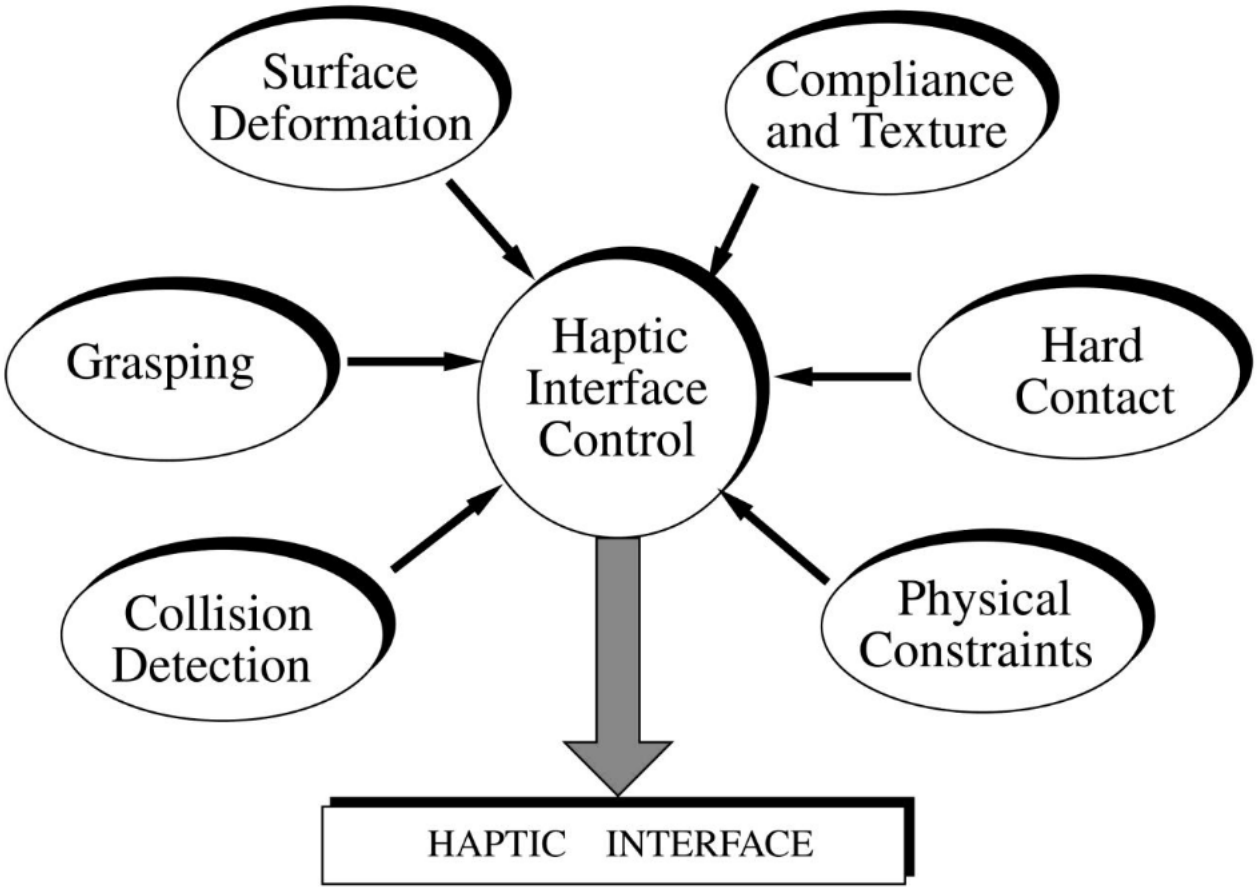
$T_{W \leftarrow car\ (t1)}$

$T_{W \leftarrow car\ (t2)}$

World system of coordinates

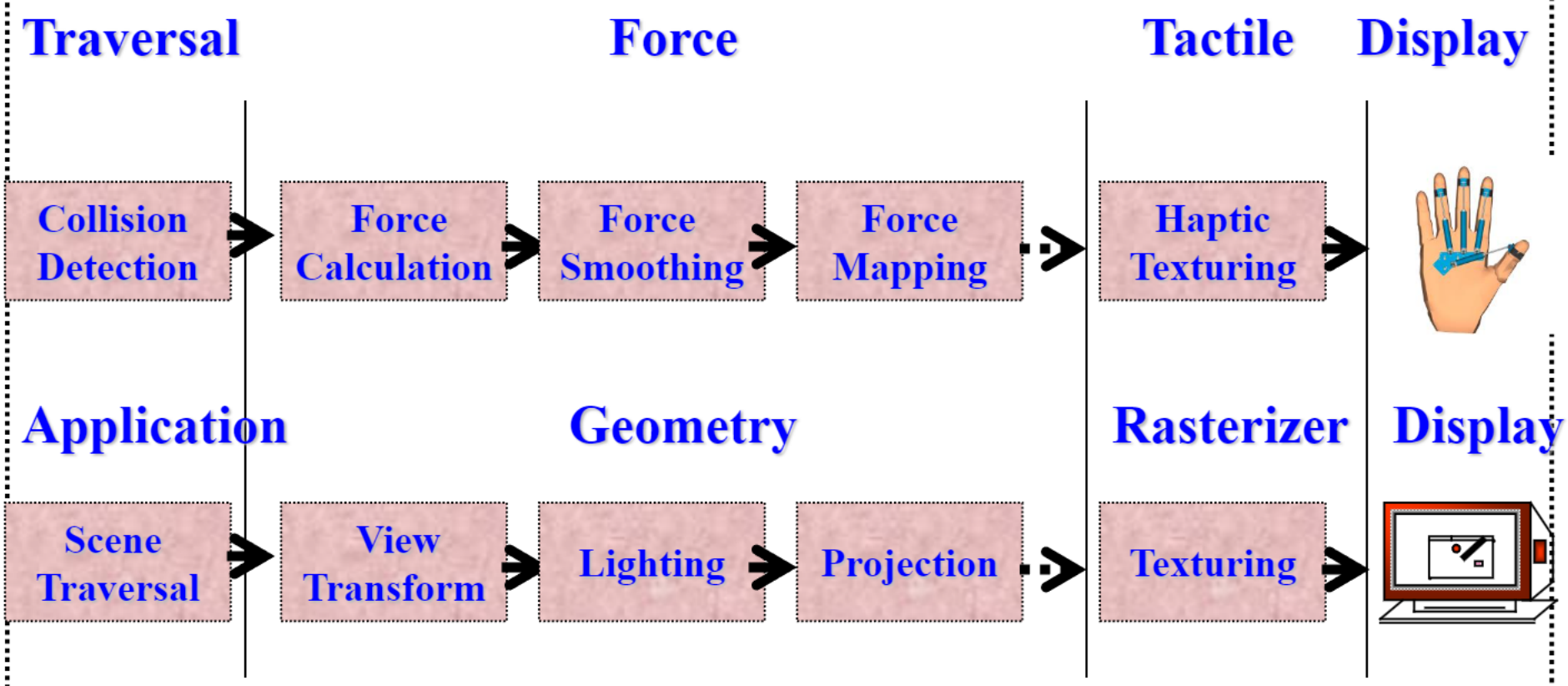
$$T_{global \leftarrow vertex(i)}(t) = T_{global \leftarrow W}(t) T_{W \leftarrow car}(t)\, V^i$$

The VR physical modeling:



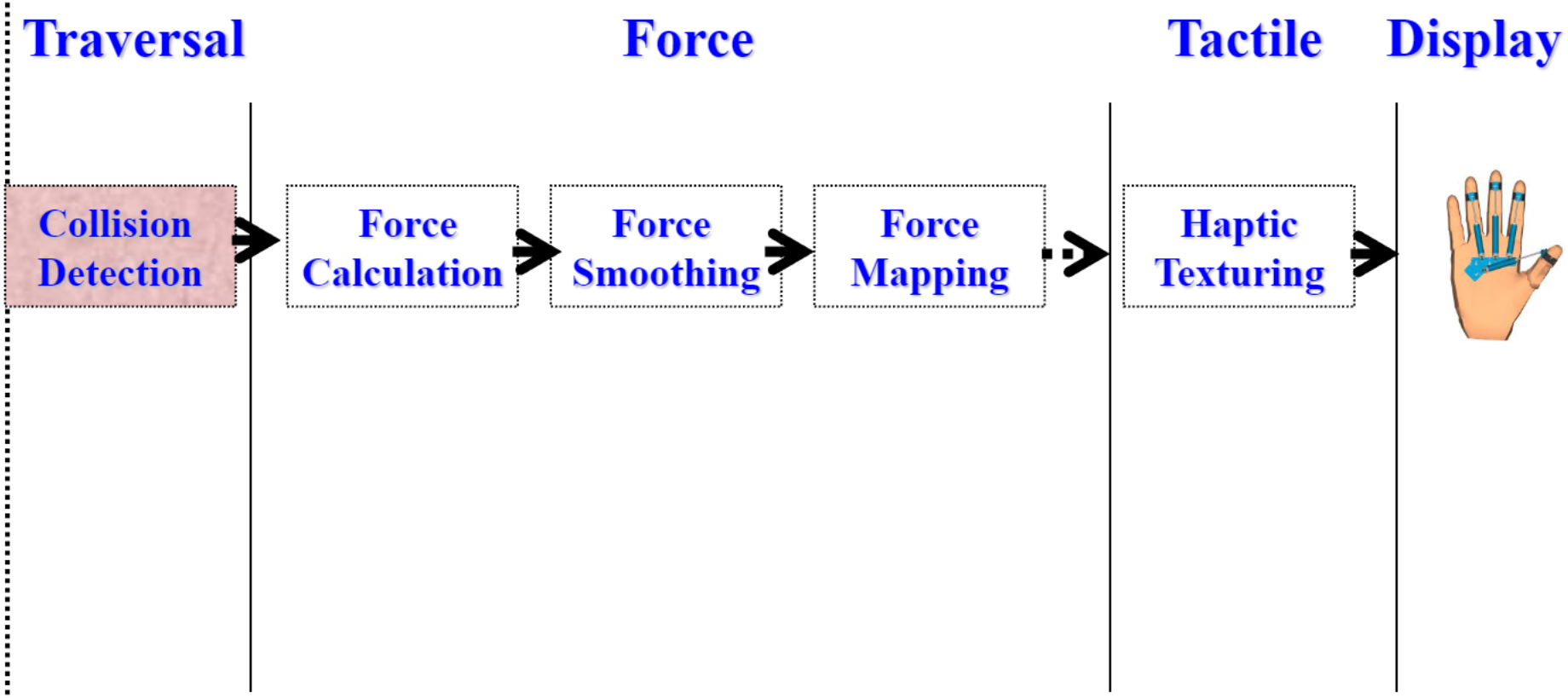
from (Burdea 1996)

The Haptics Rendering Pipeline (revisited)



adapted from (Popescu, 2001)

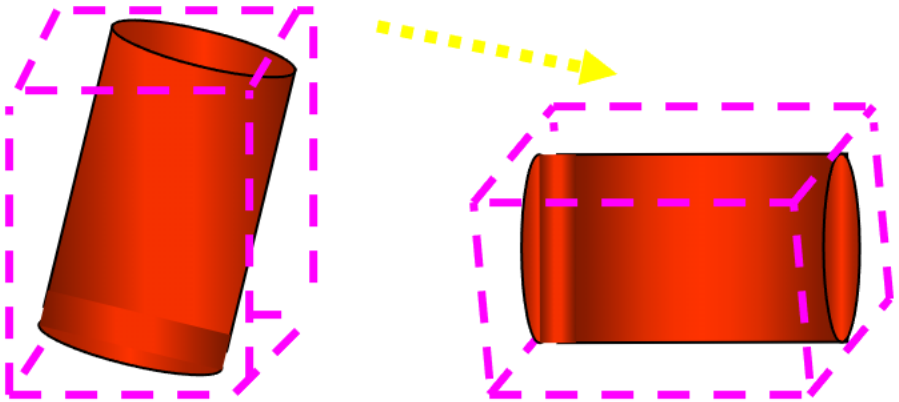
The Haptics Rendering Pipeline



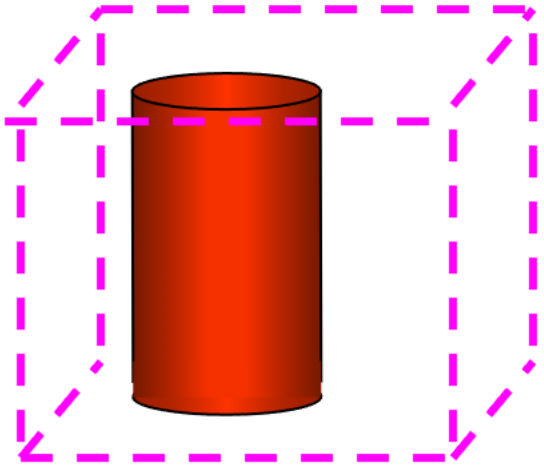
Collision detection:

- ✓ Uses ***bounding box*** collision detection for fast response;
- ✓ Two types of bounding boxes, with fixed size or variable size (depending on enclosed object orientation).
- ✓ Fixed size is computationally faster, but less precise

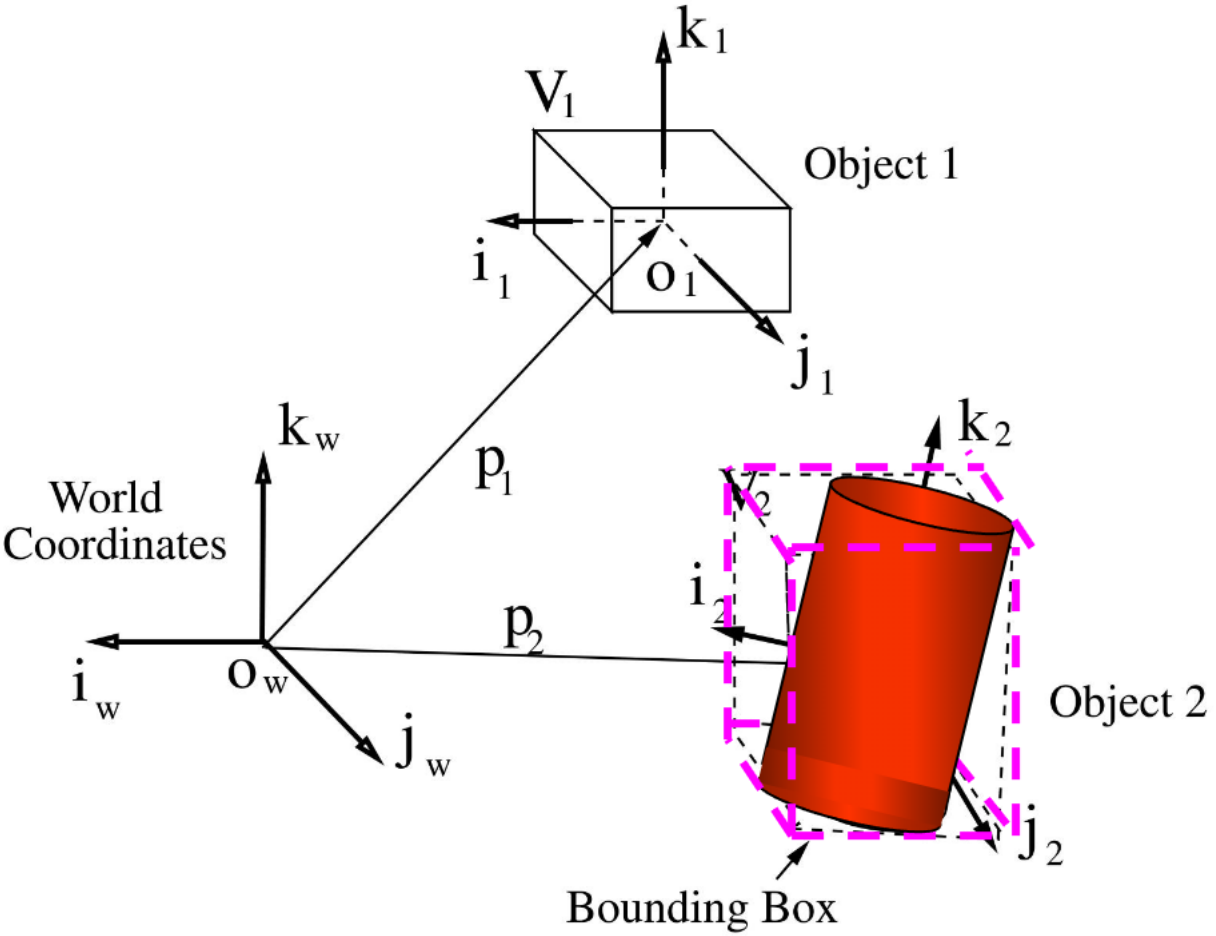
Variable size Bounding Box



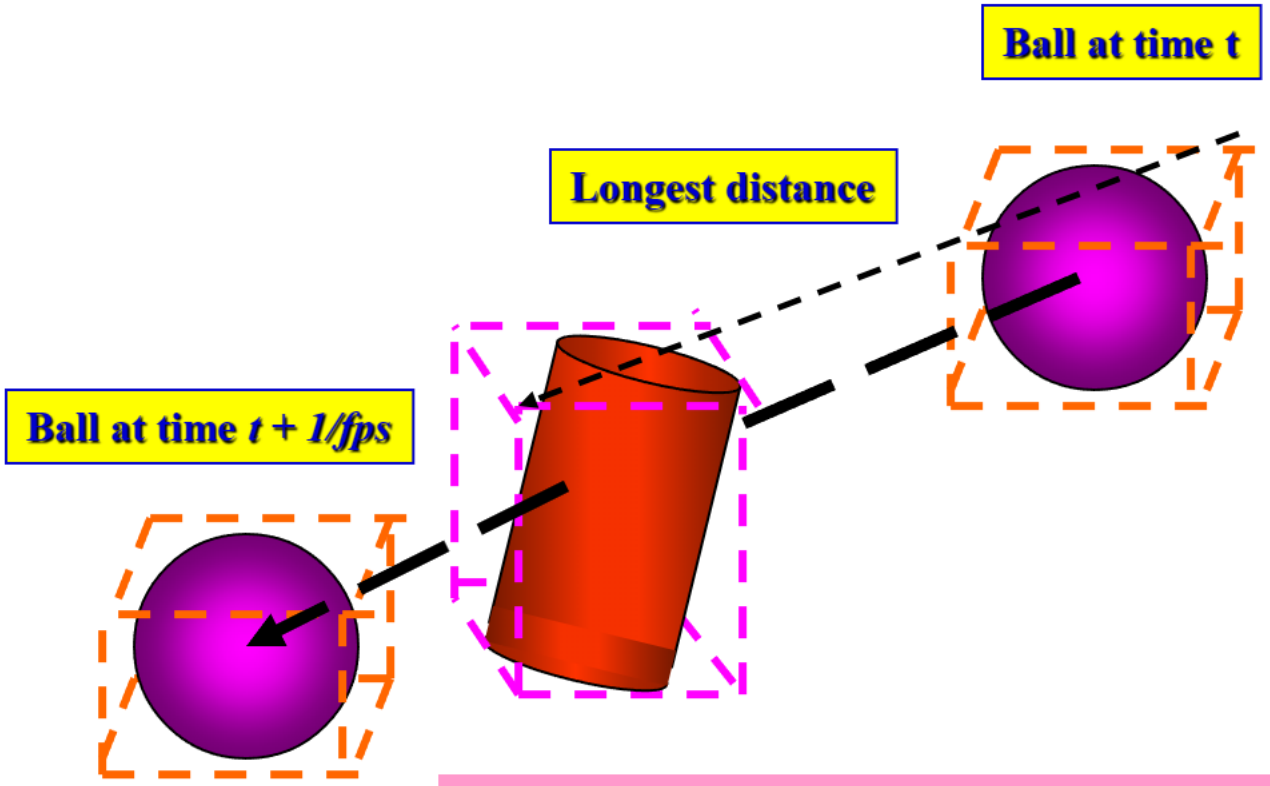
Fixed size Bounding Box



Collision Detection



Undetected collision

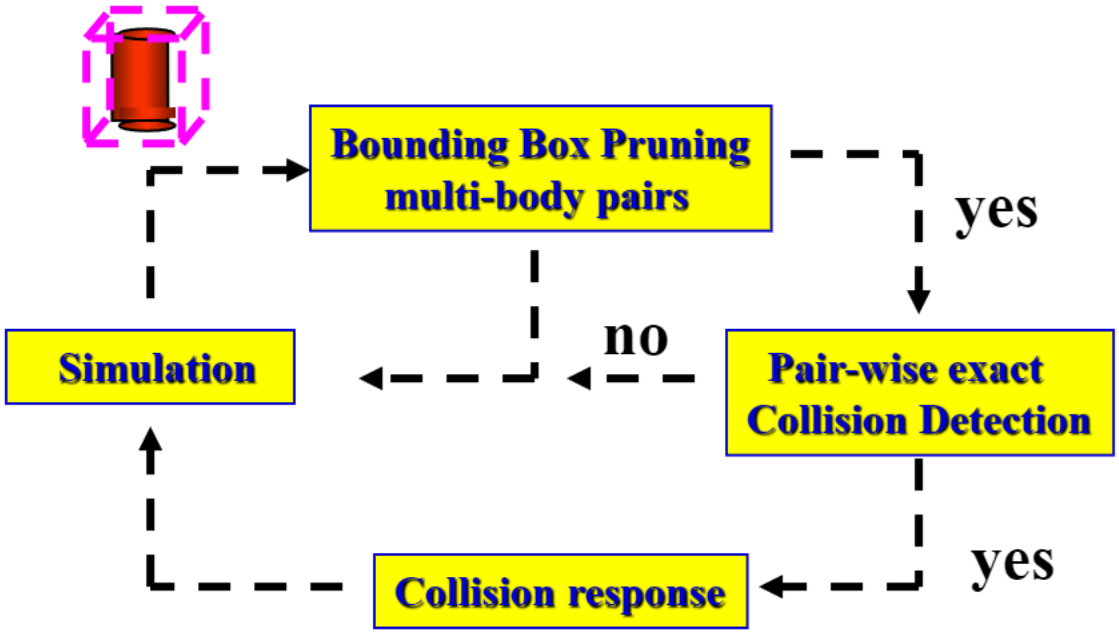


If longest distance d < v • 1/fps

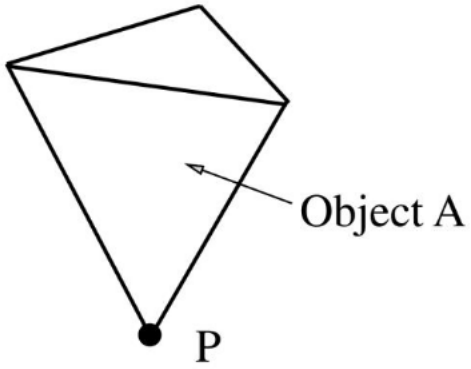
then collision is undetected (v is the ball velocity along its trajectory)

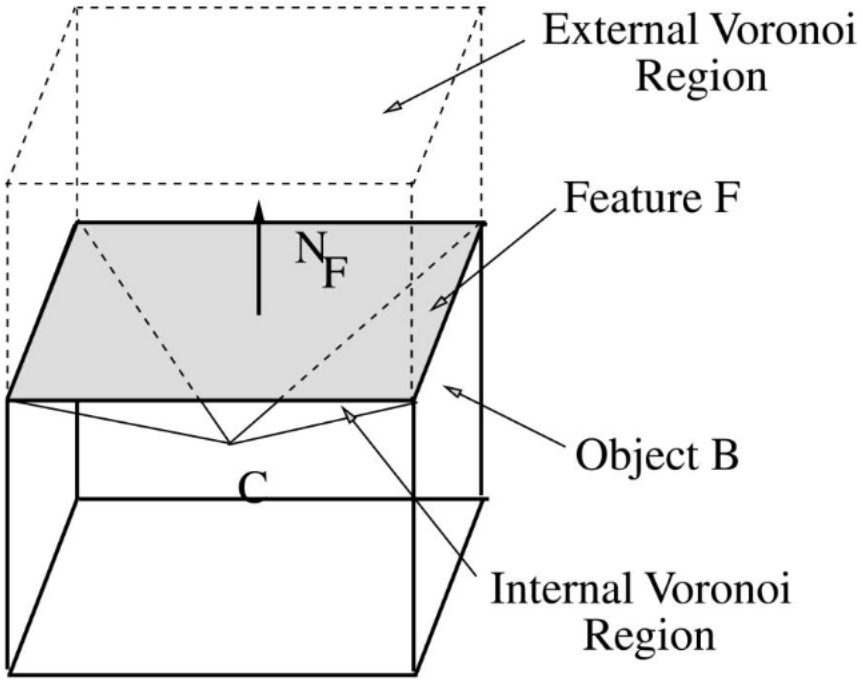
Two-stage collision detection:

✓ For more precise detection, we use a two-stage collision detection: an ***approximate*** (bounding box) stage, followed by a slower ***exact collision detection*** stage.



Exact collision detection

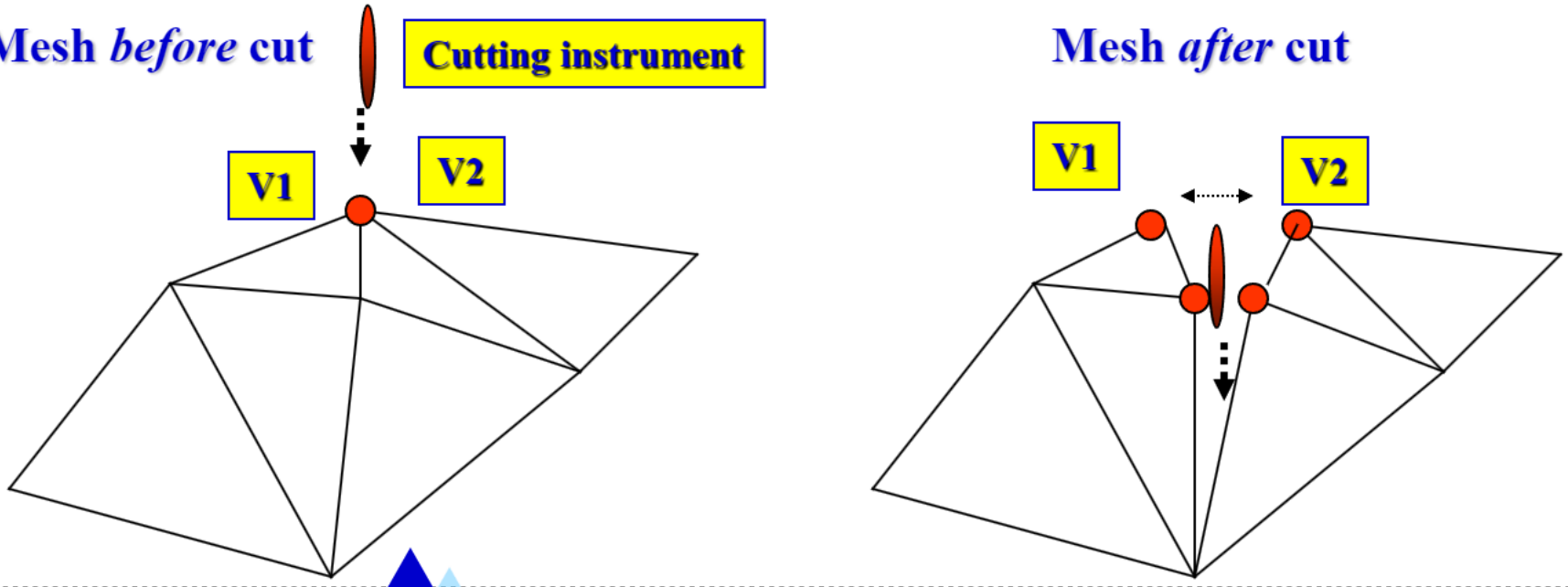




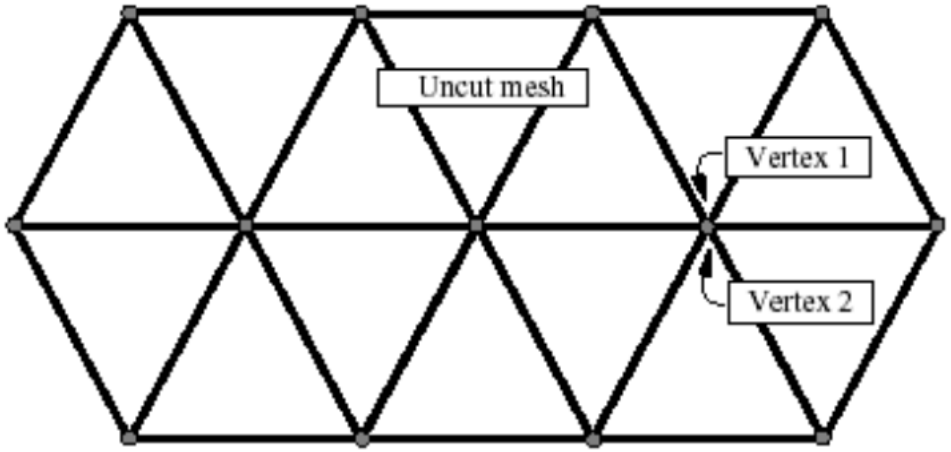
Surface cutting:

✓ An extreme case of surface "deformation" is surface cutting. This happens when the contact force exceed a given threshold;

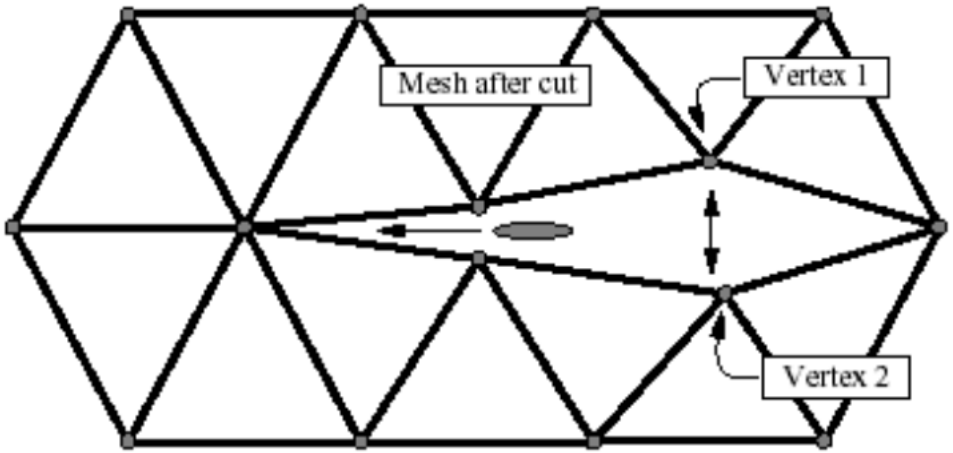
✓ When cutting, one vertex gets a ***co-located twin***. Subsequently the twin vertices separate based on spring/damper laws and the cut enlarges.

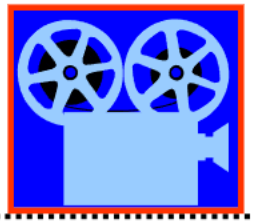


Collision response – surface deformation

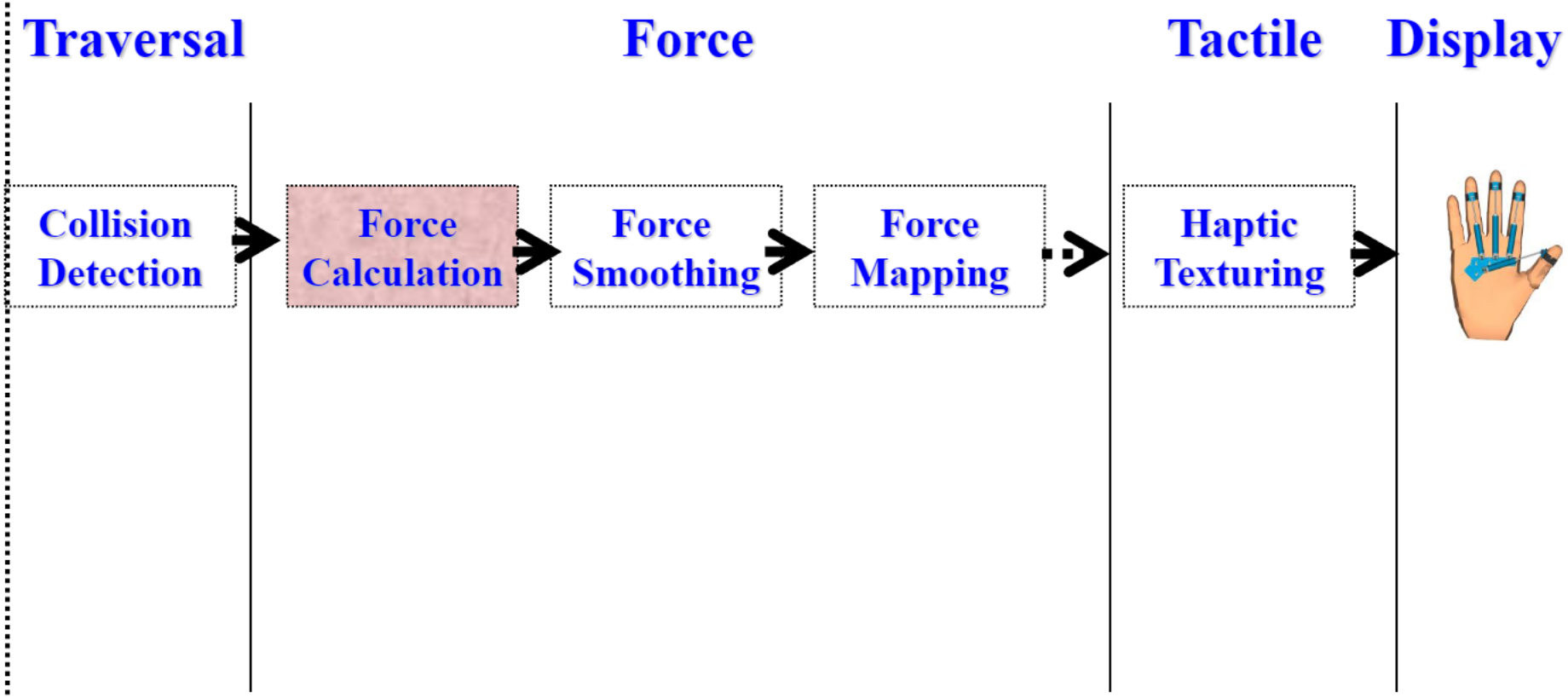


a)



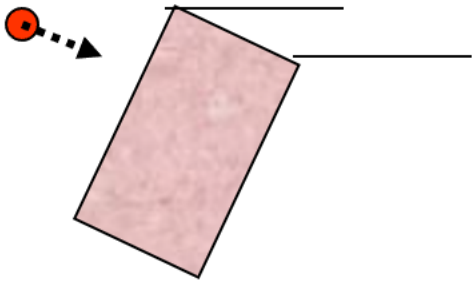


The Haptics Rendering Pipeline



Haptic interface

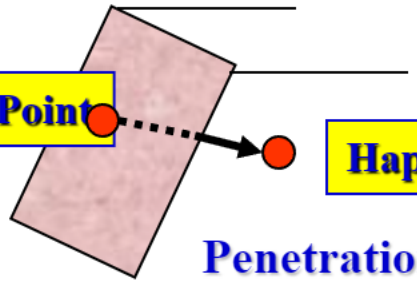
Haptic Interface Point



Object polygon

I -Haptic Interface Point

Haptic Interface Point

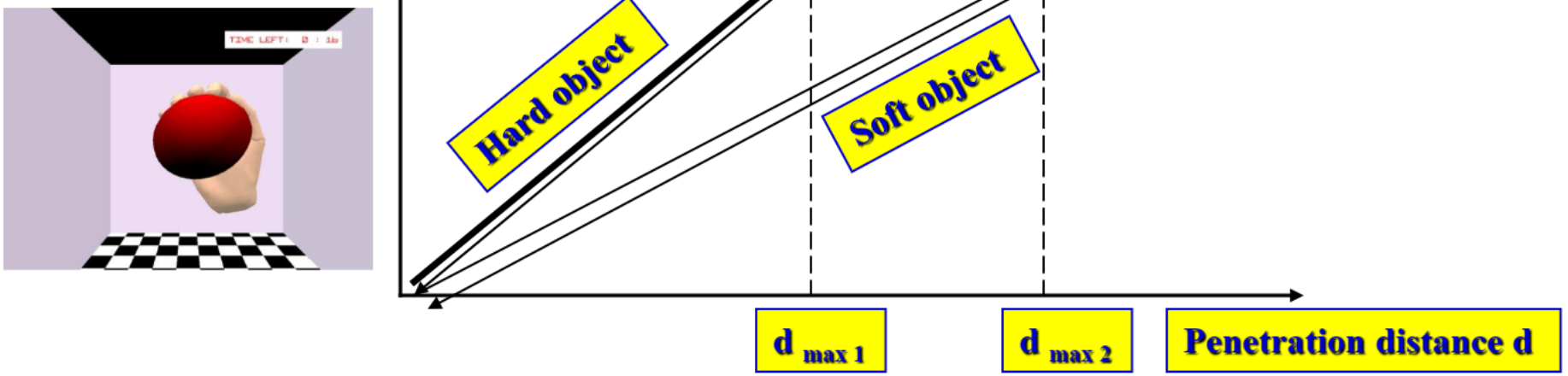


Penetration distance

Force output for homogeneous elastic objects

$$F = \begin{cases} K \bullet d, & \text{for } 0 \le d \le d_{max} \\ F_{max} & \text{for } d_{max} < d \end{cases}$$

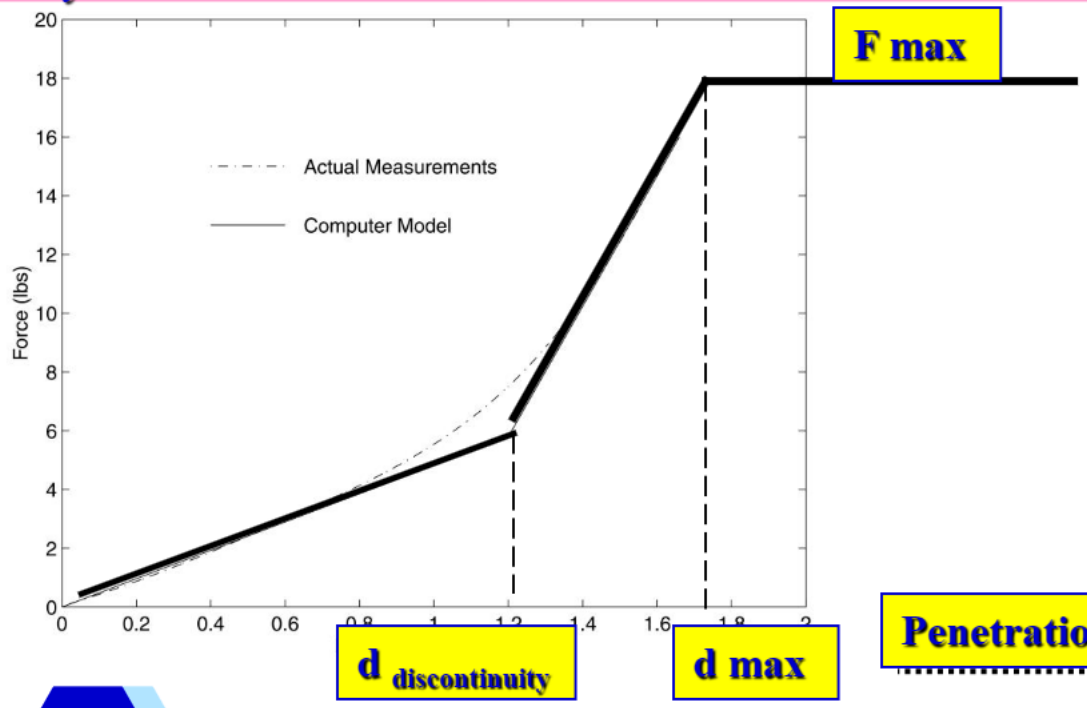
where F_{max} is that haptic interface maximum output force



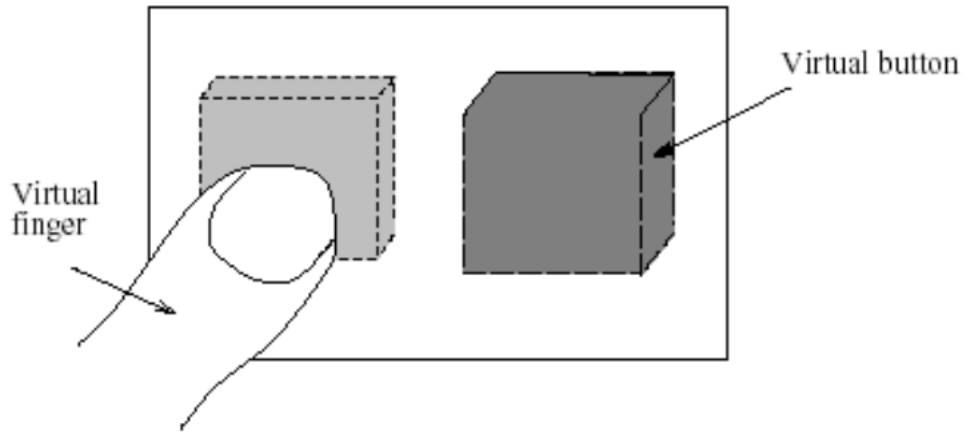
Force Calculation – Elastic objects with harder interior

$$F = \begin{cases} K_1 \cdot d, & \text{for } 0 \le d \le d_{discontinuity} \\ K_1 \cdot d_{discontinuity} + K_2 \cdot (d - d_{discontinuity}), & \text{for } d_{discontinuity} \le d \\ F_{max} & \text{for } d_{max} < d \end{cases}$$

where $d_{discontinuity}$ is object stiffness change point

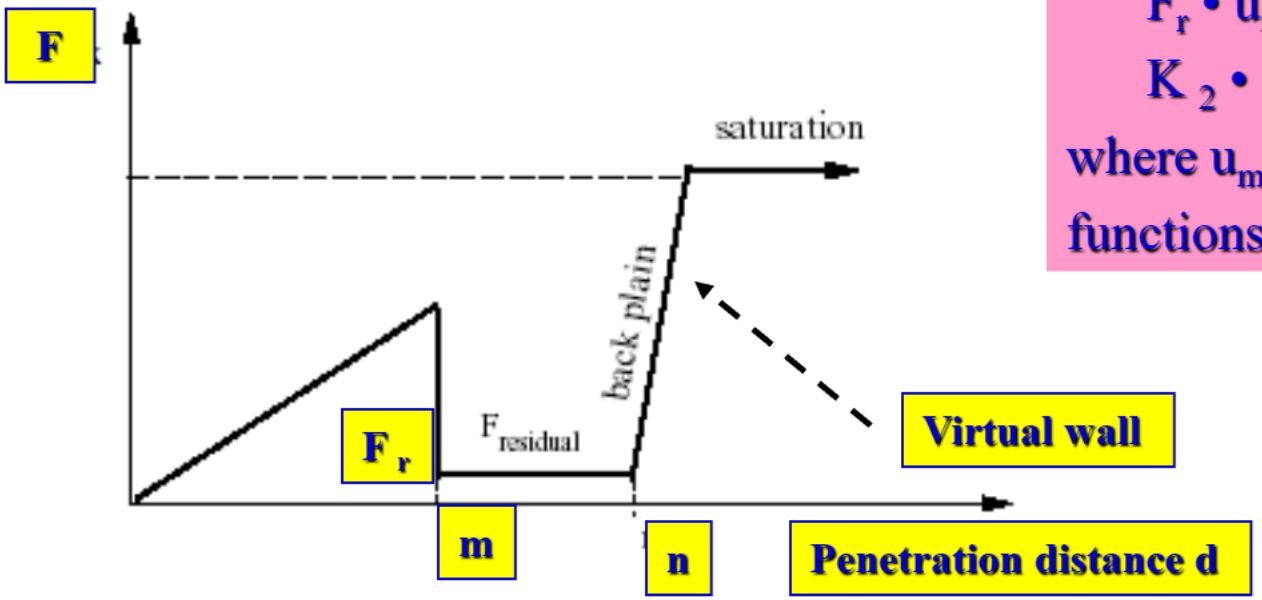


Force Calculation – Virtual pushbutton

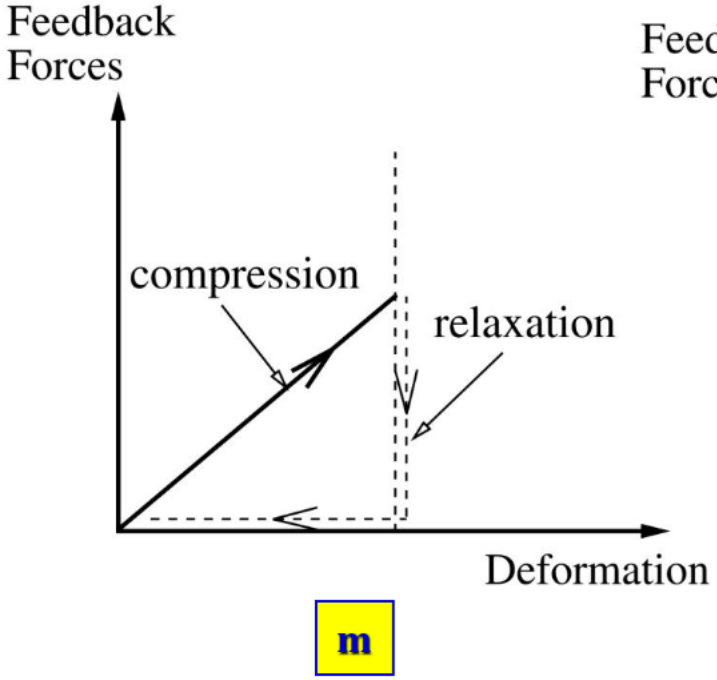


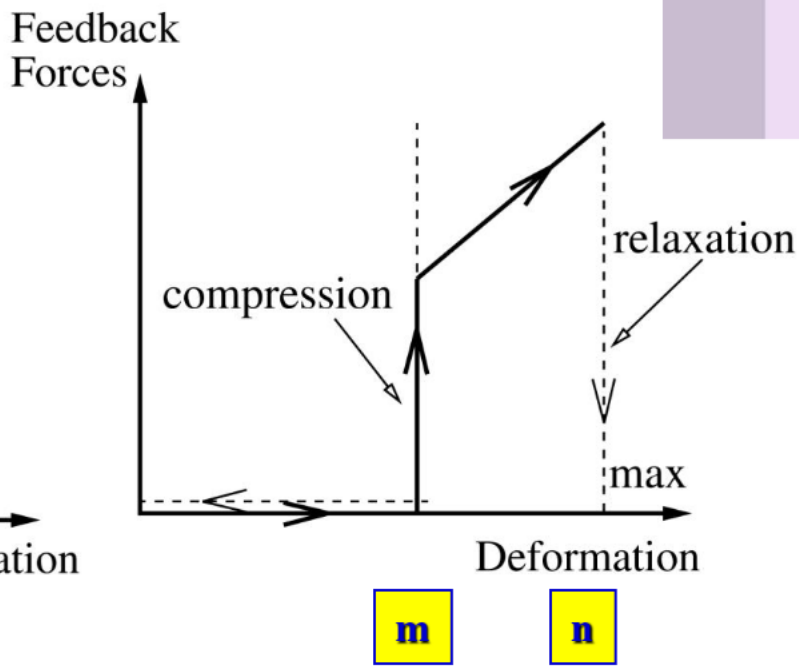
$$F = K_1 \cdot d\,(1-u_m) + F_r \cdot u_m + K_2 \cdot (d-n)\,u_n$$

where u_m and u_n are unit step functions at m and n



Force Calculation – Plastic deformation





$F_{initial} = K \bullet d$ for $0 \le d \le m$

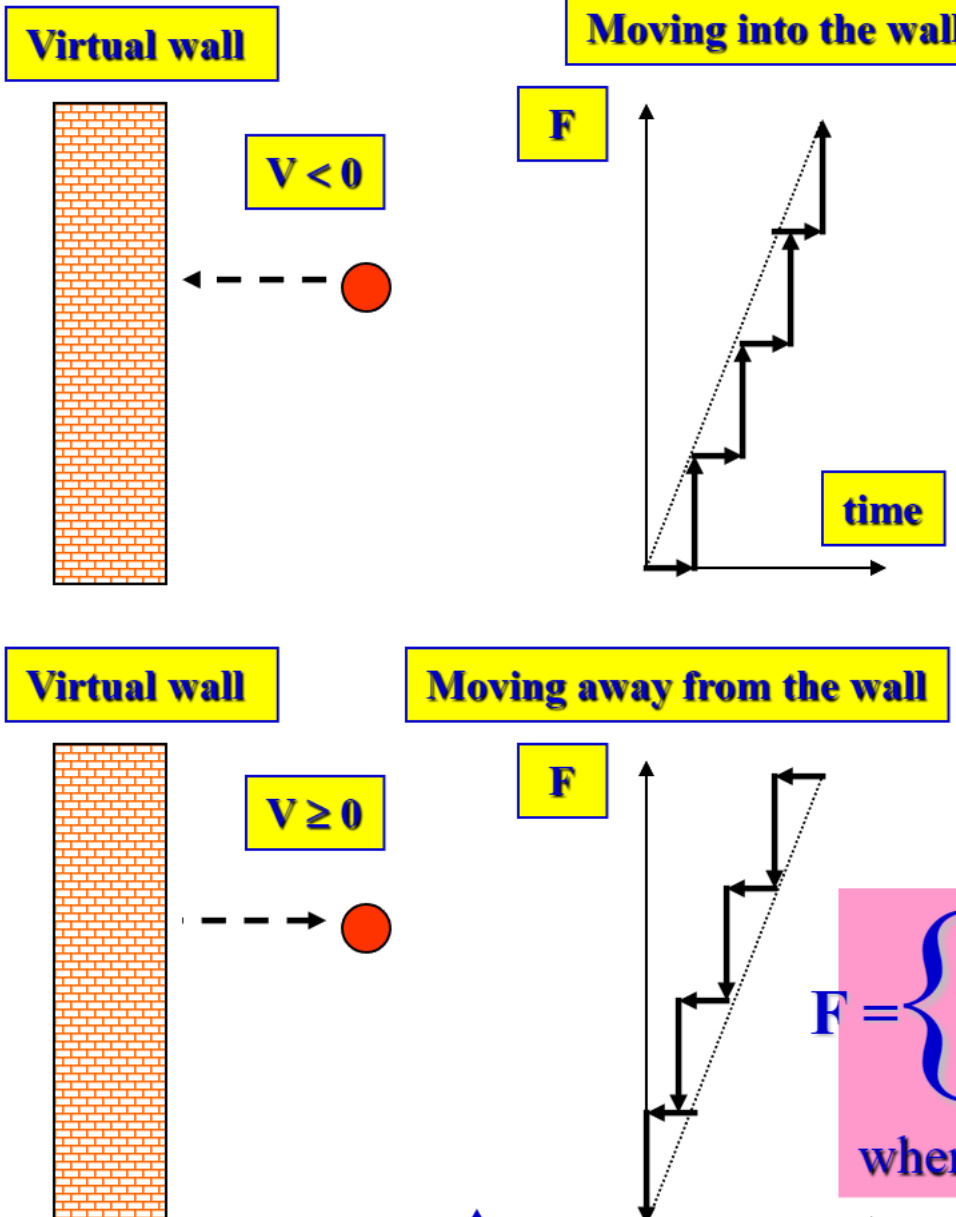
$F = 0$ during relaxation,

$F_{subsequent} = K_1 \bullet d \bullet u_m$ for $0 \le d \le n$

$F = 0$ during relaxation,

where u_m is unit step function at m

Force Calculation – Virtual wall

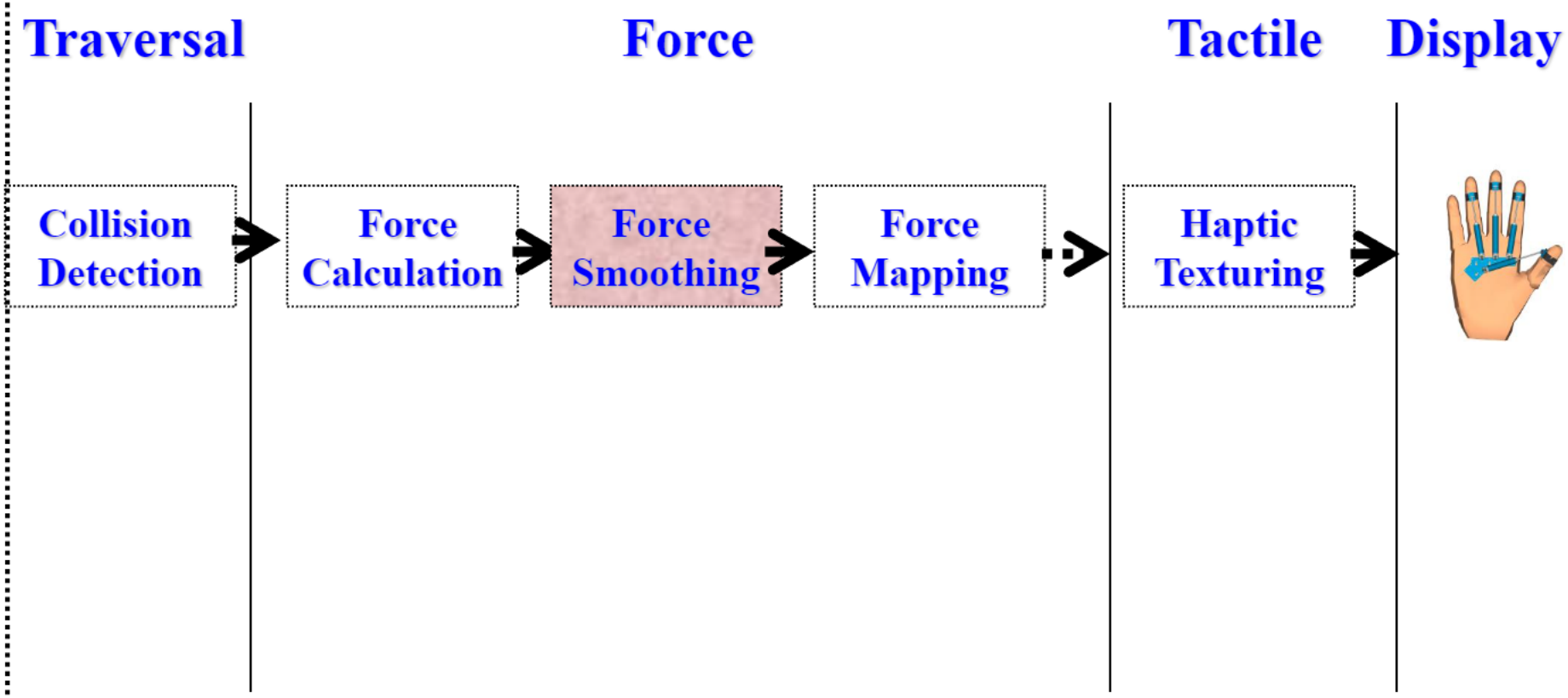


Generate energy due to sampling time- To avoid system instabilities we add a damping term

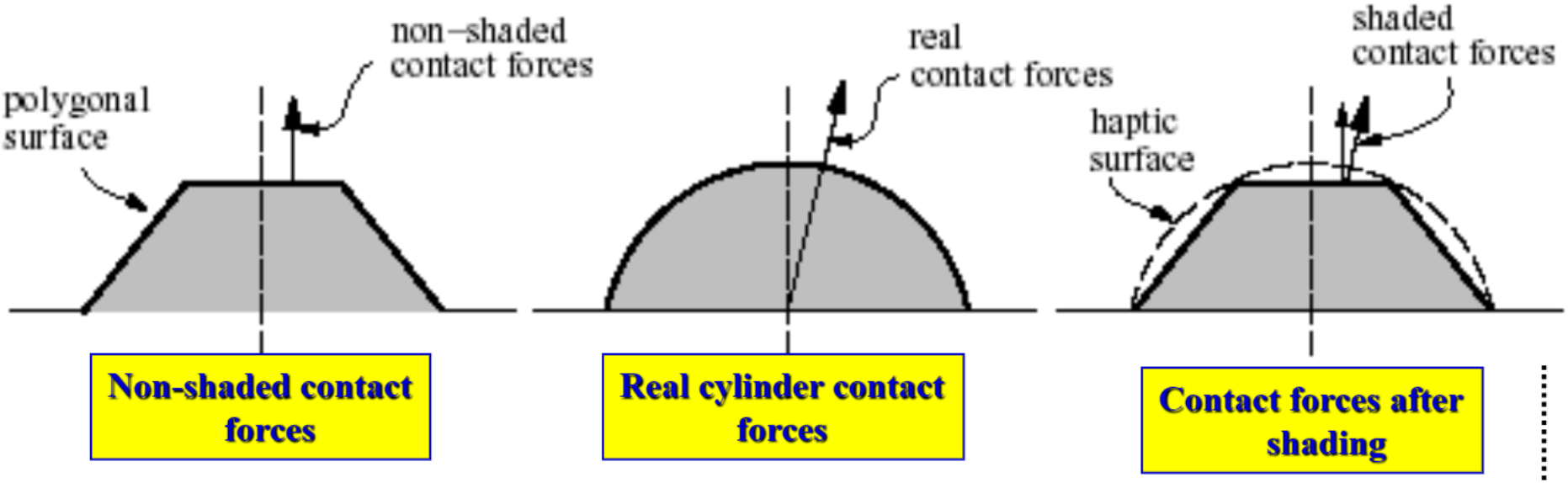
$$F = \begin{cases} K_{wall} \cdot \Delta x + B\, v, & \text{for } v < 0 \\ K_{wall} \cdot \Delta x, & \text{for } v \geq 0 \end{cases}$$

where B is a directional damper

The Haptics Rendering Pipeline



Force shading:



$$F_{smoothed} = \begin{cases} K_{object} \bullet d \bullet \overline{N}, & \text{for } 0 \le d \le d_{max} \\ F_{max} \bullet \overline{N}, & \text{for } d_{max} < d \end{cases}$$

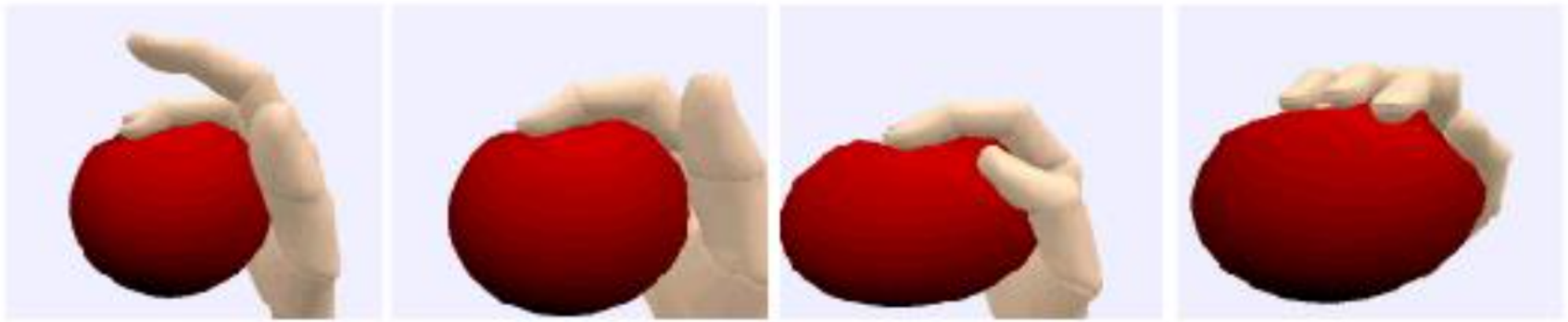
where $\overline{N}$ is the direction of the contact force based on vertex normal interpolation

The haptic mesh:

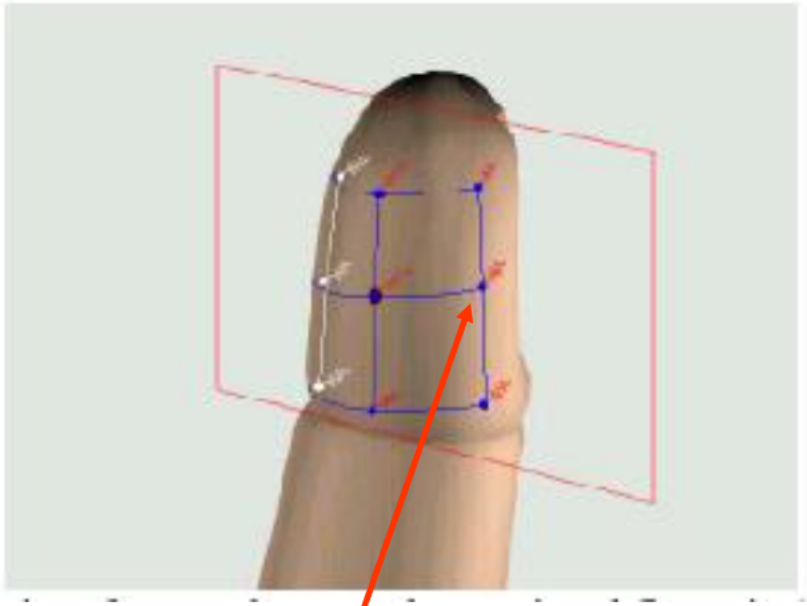
✓ A single HIP (haptic interface point) is not sufficient to capture the geometry of fingertip-object contact;

✓ The curvature of the fingertip, and the object deformation need to be realistically modeled.

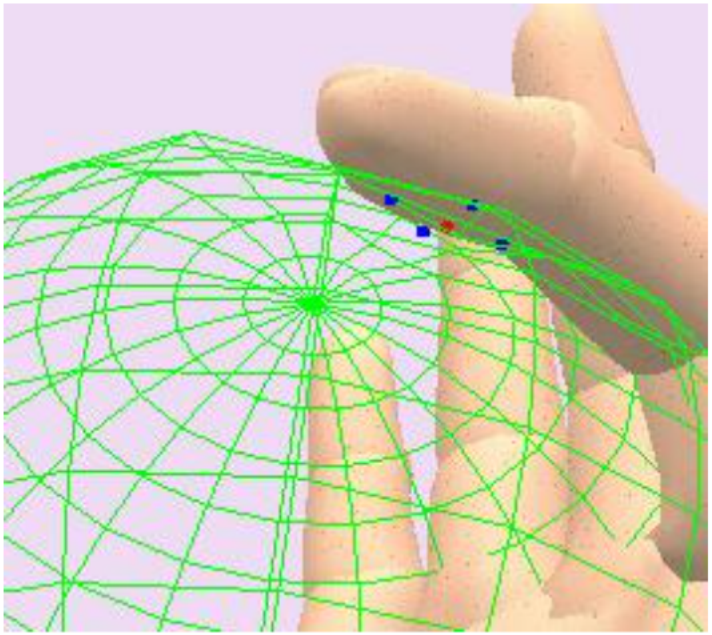
Screen sequence for squeezing an elastic virtual ball



Haptic mesh

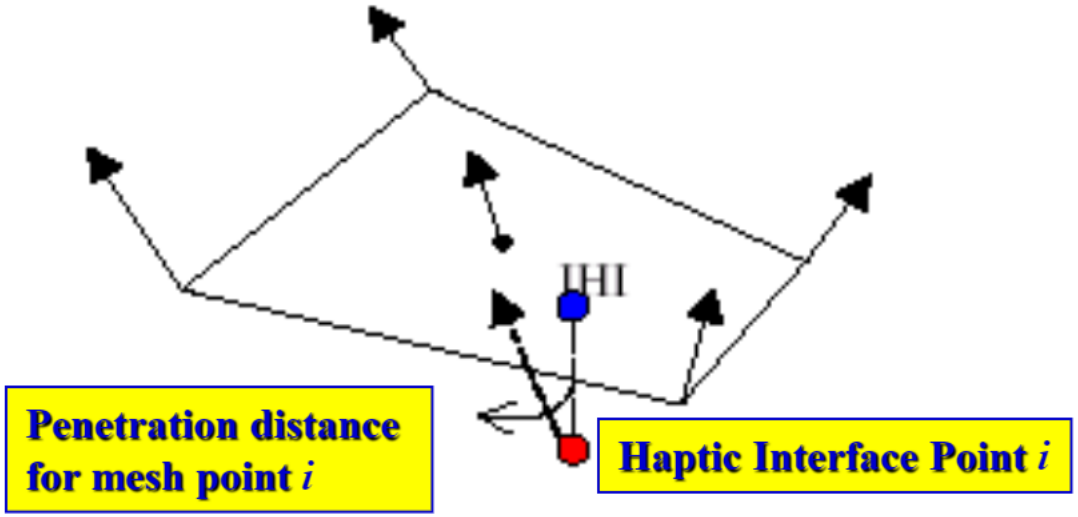


Mesh point *i*



Penetration distance for mesh point *i*

Haptic mesh force calculation

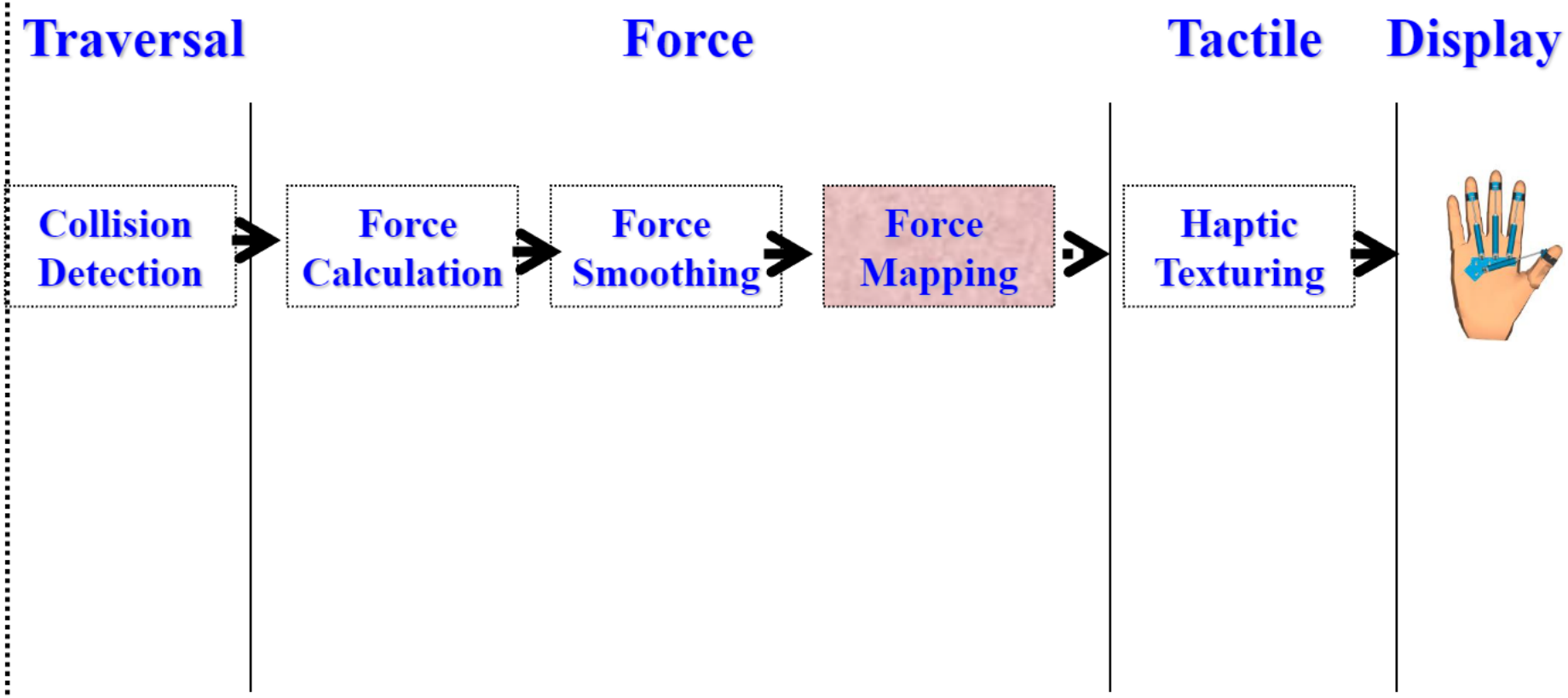


For each haptic interface point of the mesh:

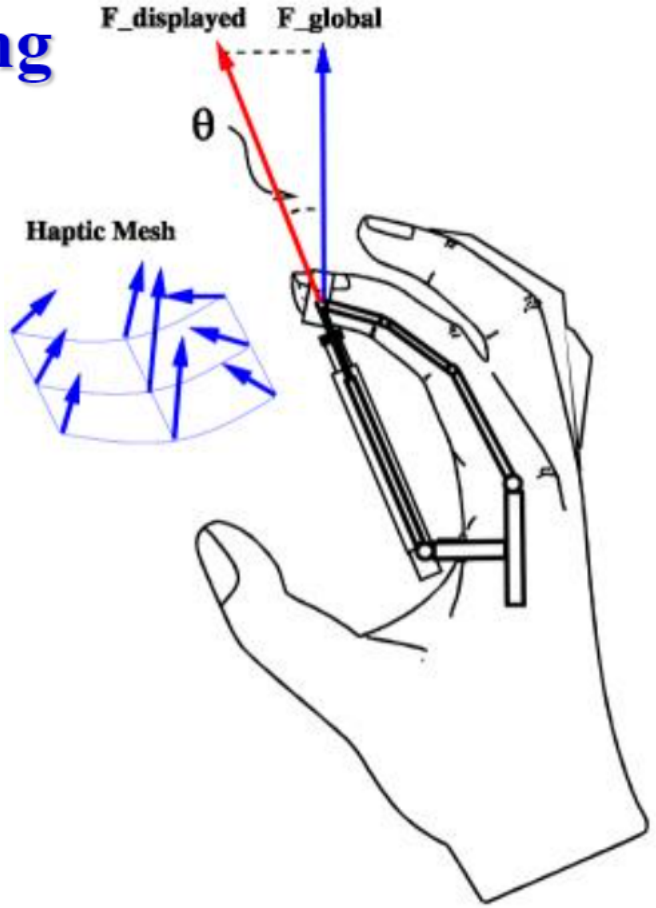
$$F_{\text{haptic-mesh i}} = K_{\text{object}} \bullet d_{\text{mesh i}} \bullet \overline{N}_{\text{surface}}$$

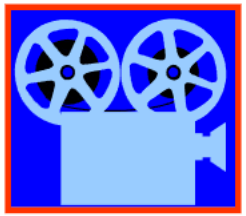
where $d_{\text{mesh i}}$ are the interpenetrating distances at the mesh points, $\overline{N}_{\text{surface}}$ is the weighted surface normal of the contact polygon

The Haptics Rendering Pipeline



Force mapping





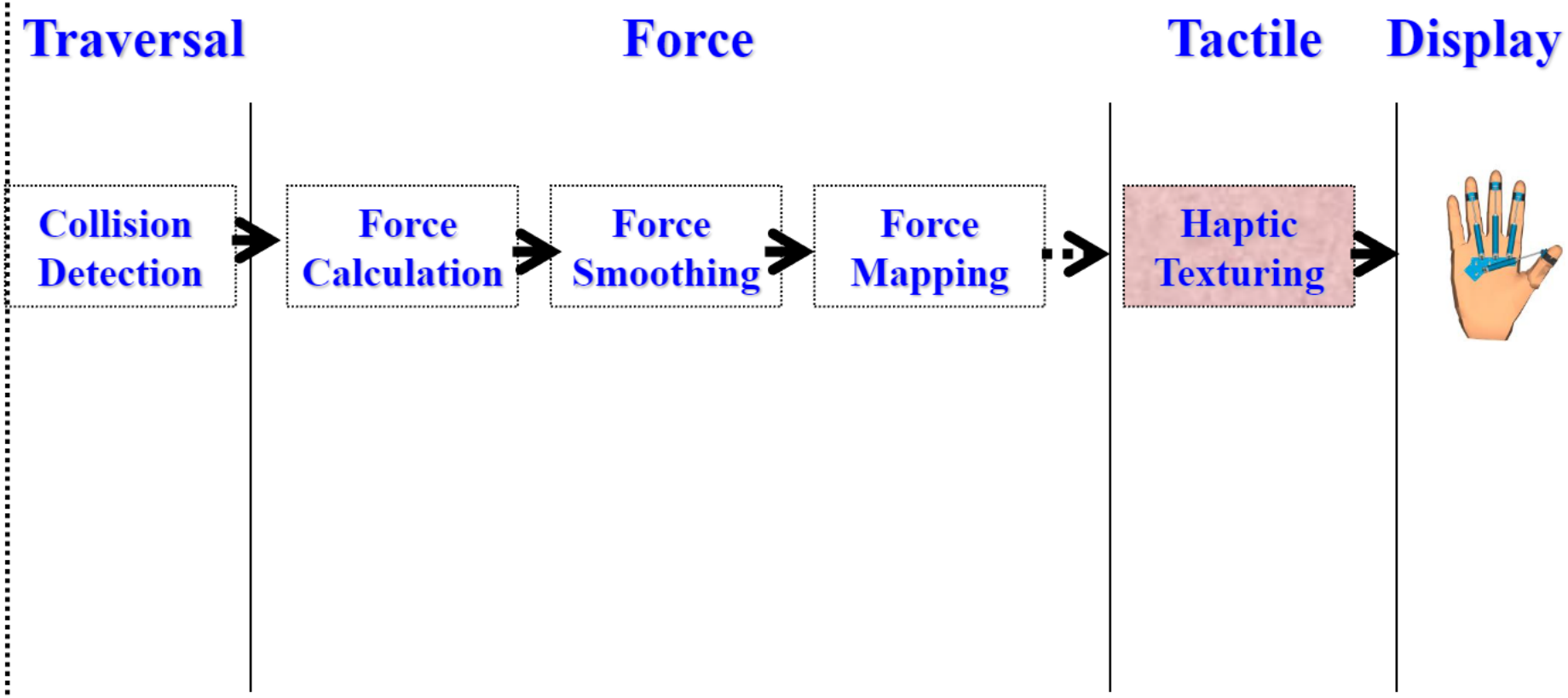
VC 5.2

Force displayed by the Rutgers Master II interface:

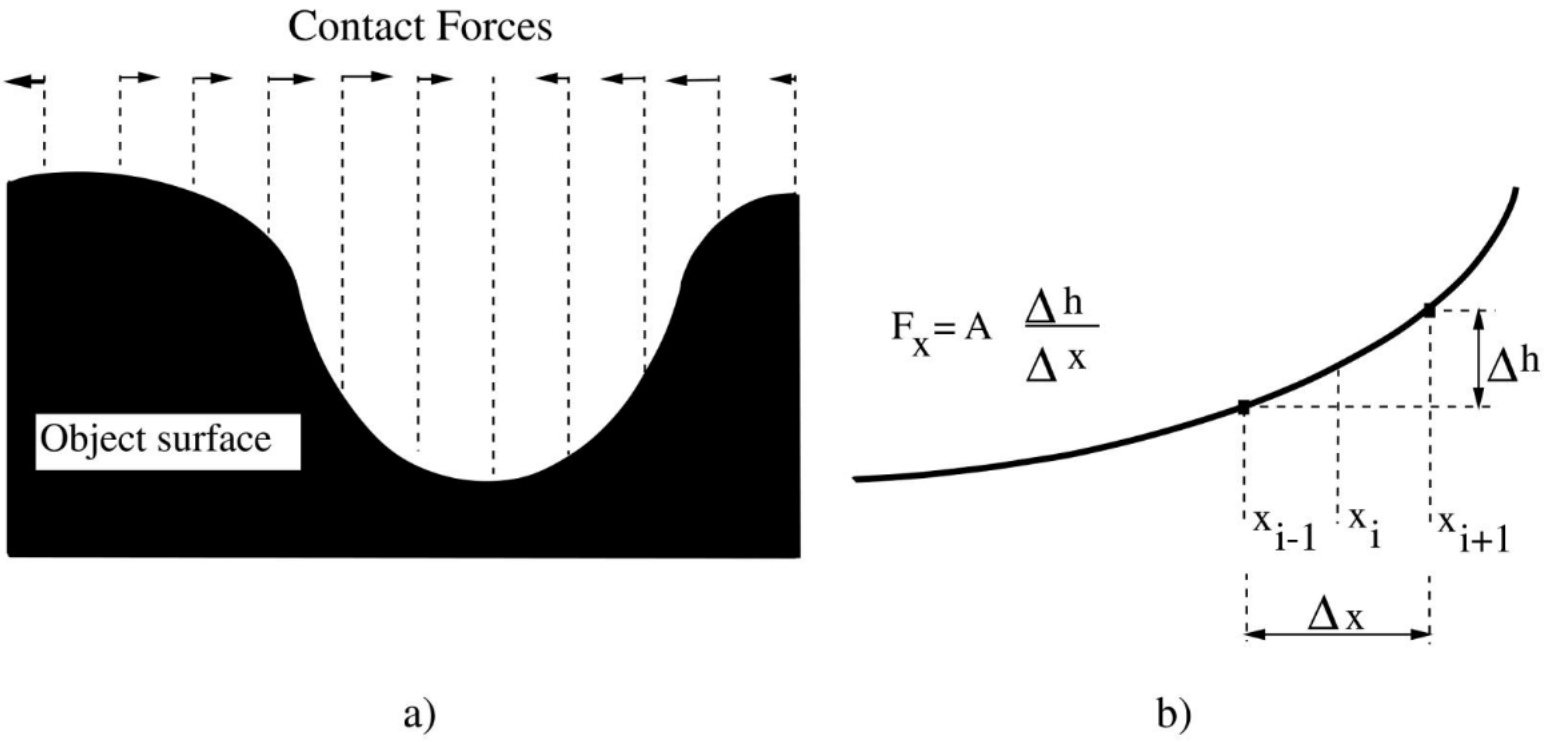
$F_{displayed} = (\Sigma F_{haptic\text{-}mesh})/\cos\theta$

where θ it the angle between the mesh force resultant and the piston

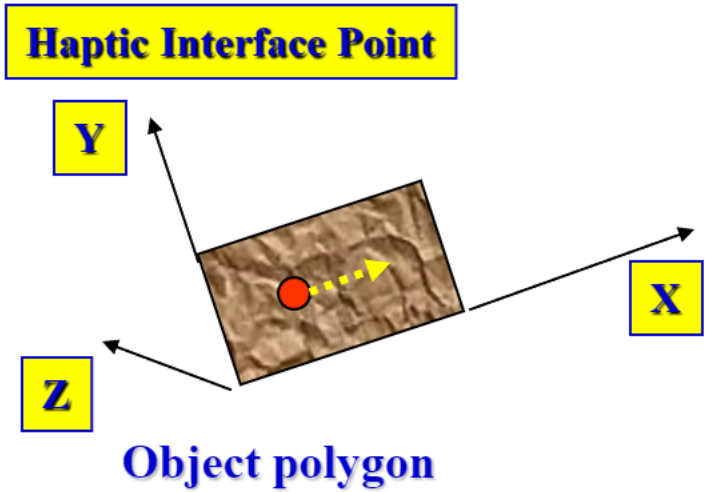
The Haptics Rendering Pipeline



Surface haptic texture produced by the PHANToM interface



Haptic interface



$F_{texture} = A \sin(m\,x) \cdot \sin(n\,y)$,

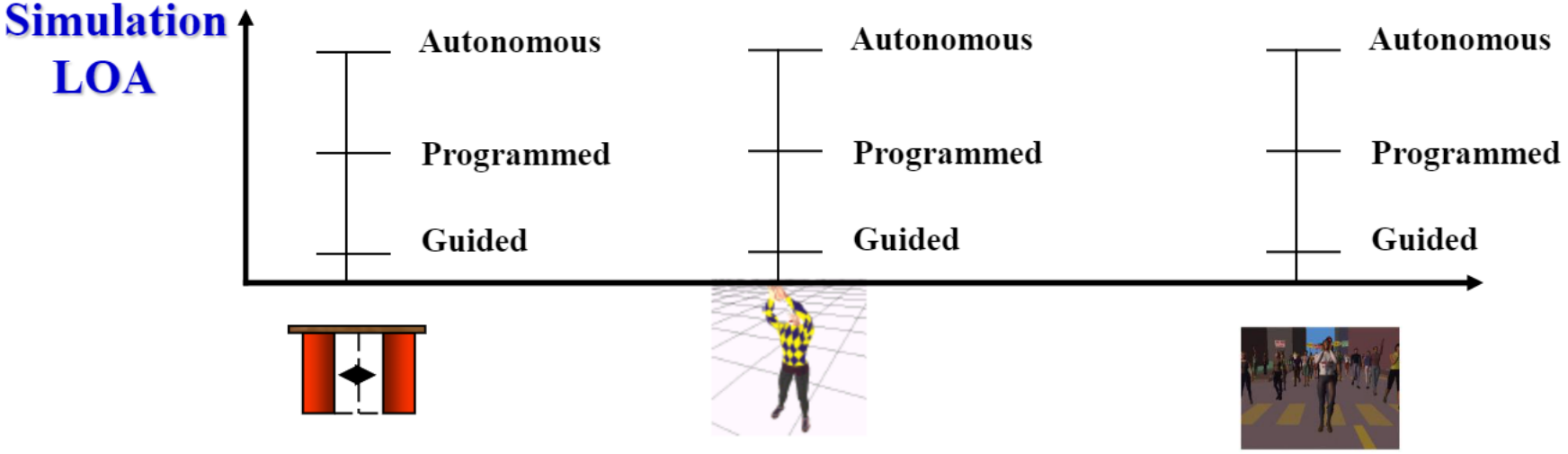
where A, m, n are constants:

- A gives magnitude of vibrations;
- m and n modulate the frequency of vibrations in the x and y directions

BEHAVIOR MODELING

- ✓ The simulation level of autonomy (LOA) is a function of its components
- ✓ Thalmann et al. (2000) distinguish three levels of autonomy. The simulation components can be either "guided" (lowest), "programmed" (intermediate) and "autonomous (high)

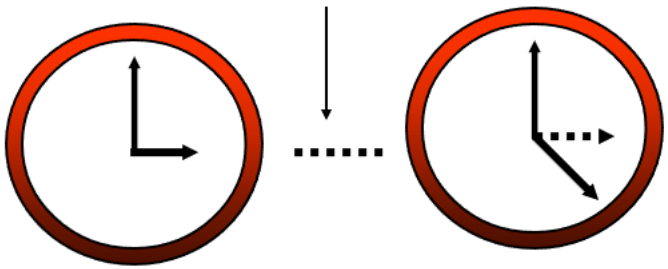
Simulation LOA = f(LOA(Objects),LOA(Agents),LOA(Groups))

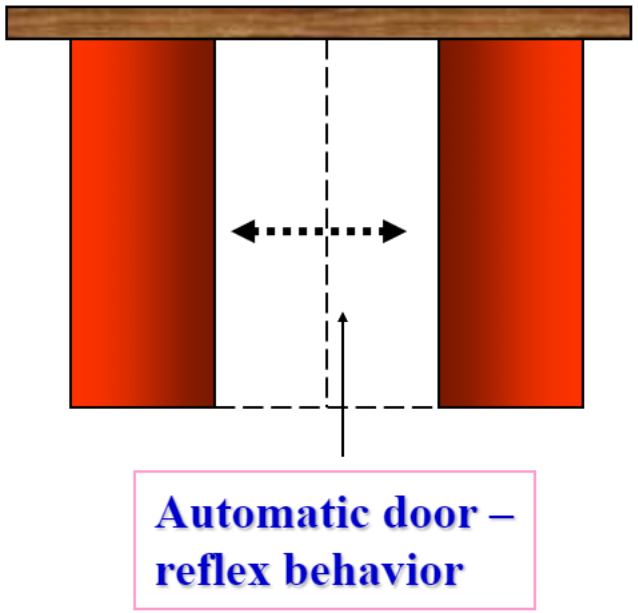


adapted from (Thalmann et al., 2000)

Interactive objects:

- ✓ Have behavior independent of user's input (ex. clock);
- ✓ This is needed in large virtual environments, where it is impossible for the user to provide all required inputs.



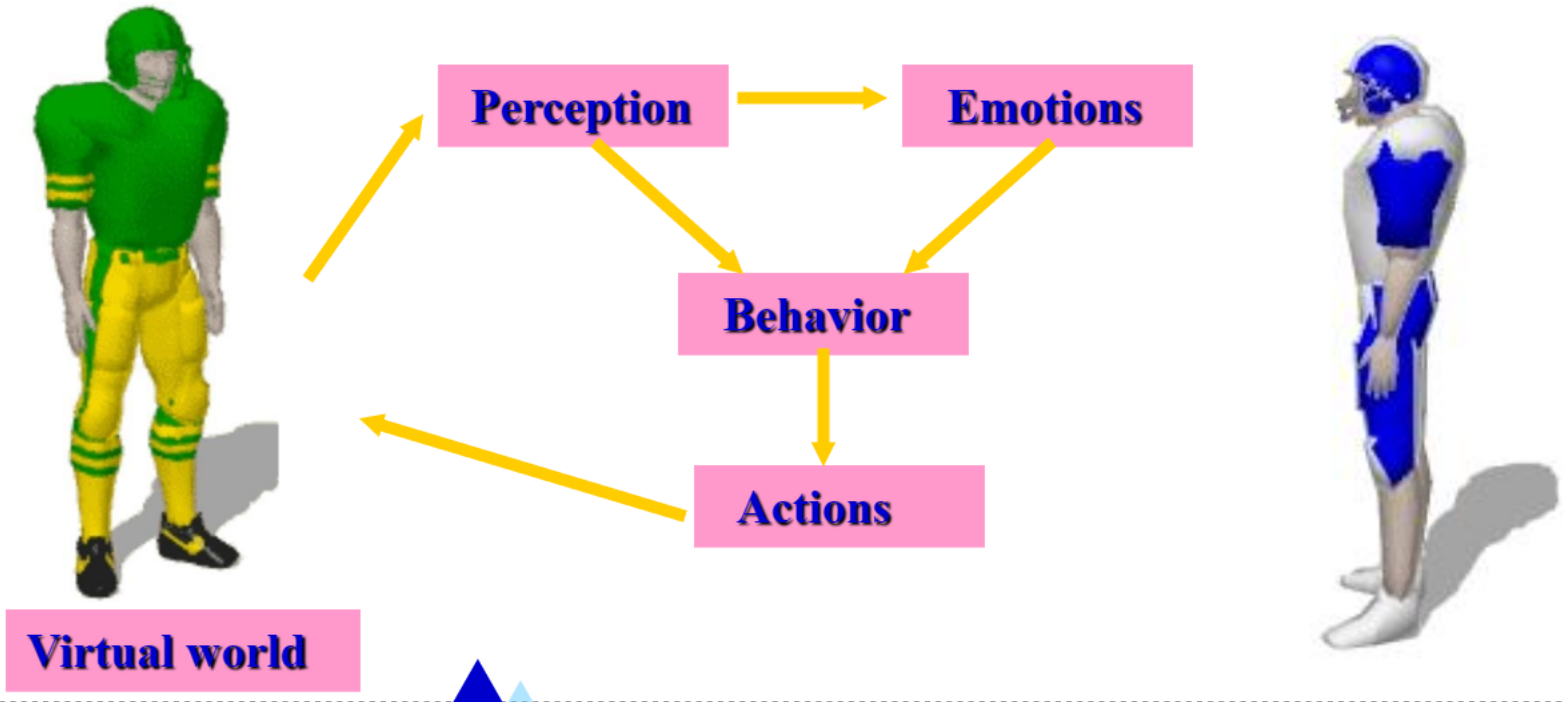


Interactive objects:

- ✓ The fireflies in NVIDIA's ***Grove*** have behavior independent of user's input. User controls the virtual camera;

Agent behavior:

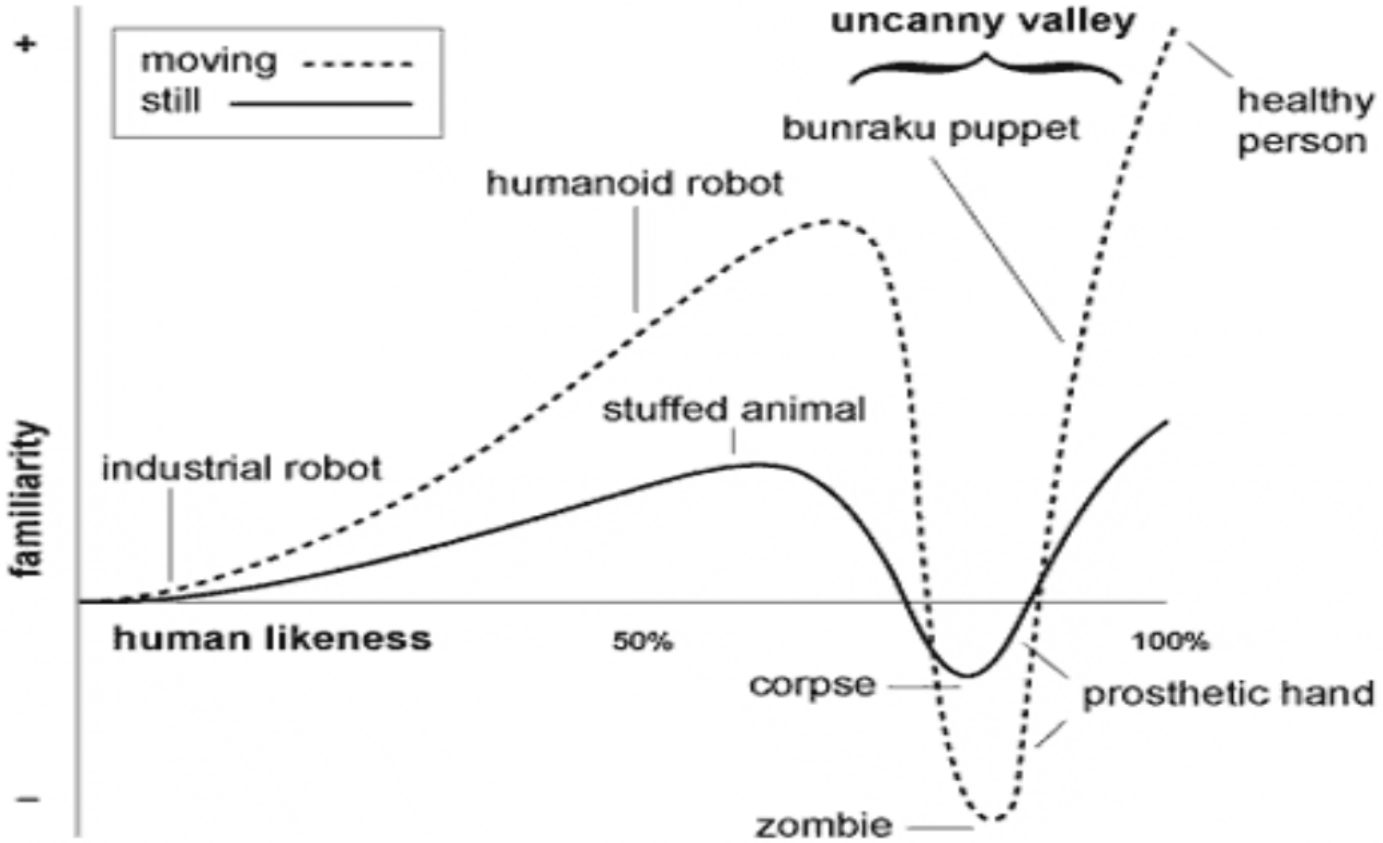
- ✓ A behavior model composed of ***perception***, ***emotions***, ***behavior***, and ***actions***;
- ✓ Perception (through virtual sensors) makes the agent aware of his surroundings.



Avatar

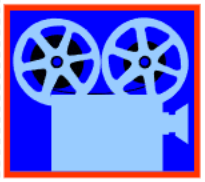
- Written in 1994
  - Development Inspired By Lord of the Rings
- Goal: animation technology that allows for close-up shots surpassing uncanny valley

The Uncanny Valley



Avatar Technology

- Full Body Motion Capture
  - Used for both actors and other in world objects as stand ins
- Facial Motion Capture
  - Imparts human emotion to facial animation
- Director's Camera
  - Low resolution view of virtual world



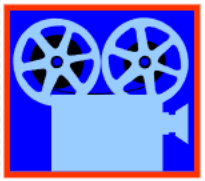
Facial Capture


12.1mm i0.00 c0.0
195_tk_00E_002_Z1_pc001_0A01_VC_Av001
00:00:22.17 13:52:22.12 13:52:22.10
545 767 767
LN 08 0707-4713+12


13.8mm i0.00 c0.0

L.A. Noir

- 1940's Detective Game
  - Witness interrogation
- Incorporates Facial motion capture technology
  - Does not require facial markers

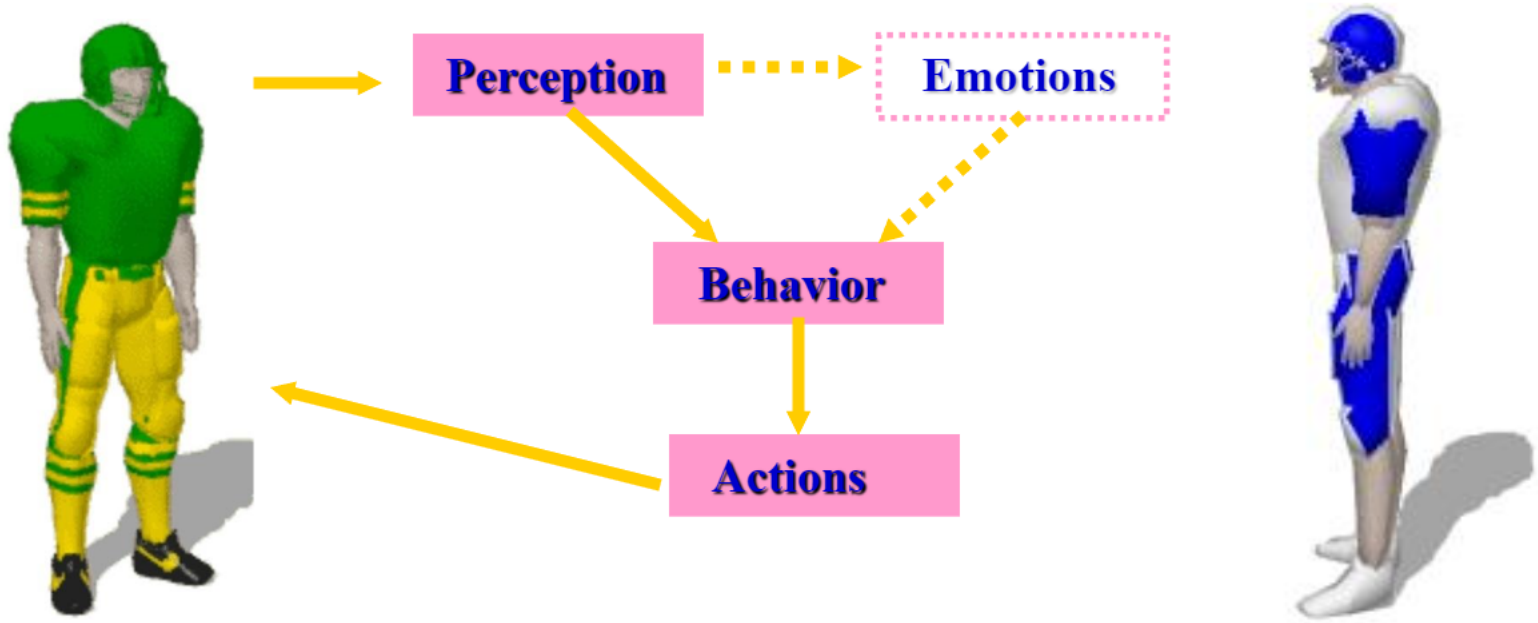


References

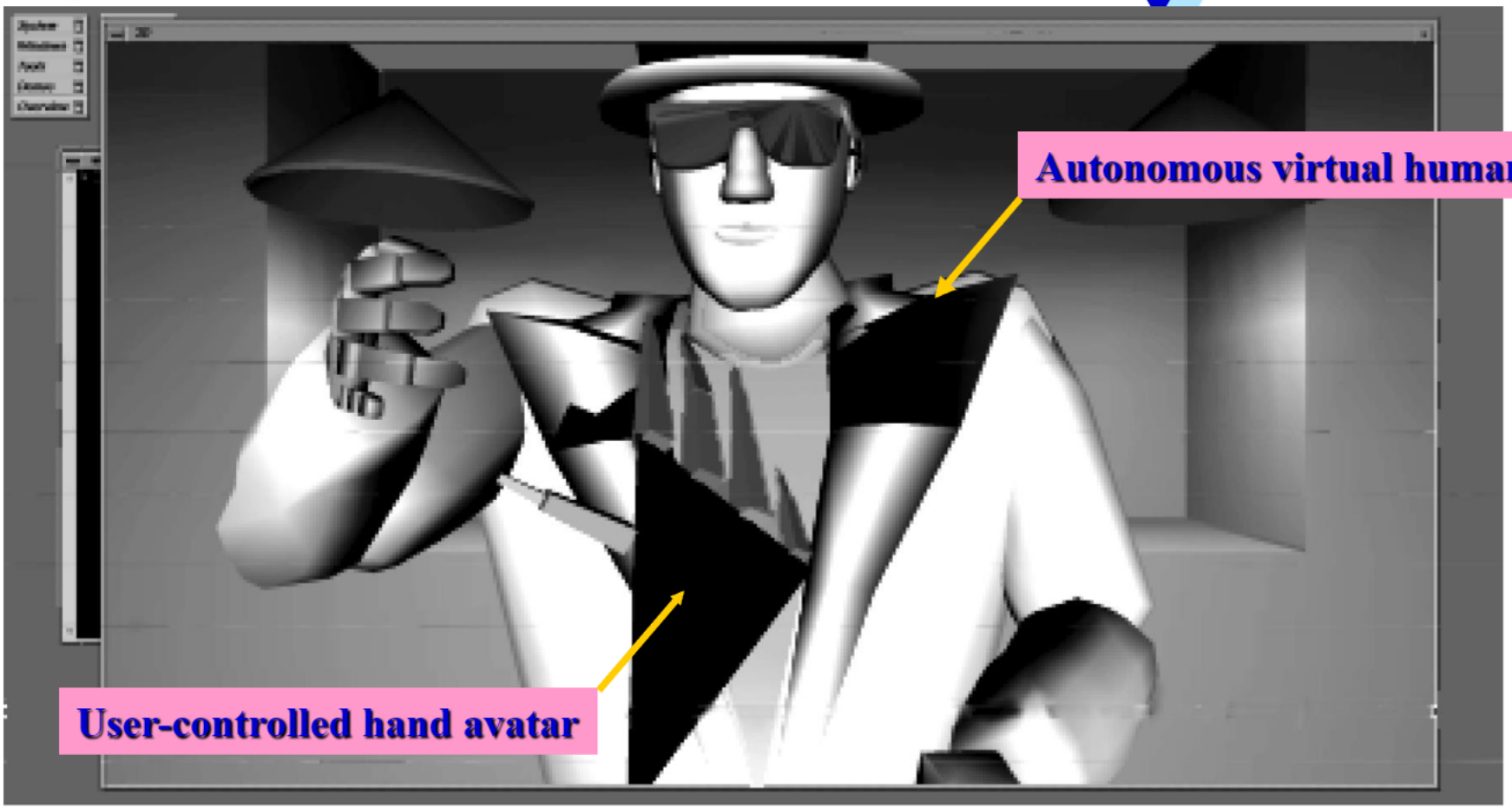
- http://www.gametrailers.com/video/developer-diary-l-a-noire/708504
- http://www.youtube.com/watch?v=O6OKMhSlSIY&feature=related
- http://www.moviemobsters.com/2009/07/18/monster-madness-creature-4/
- http://en.wikipedia.org/wiki/Avatar_%282009_film%29
- http://en.wikipedia.org/wiki/L.A._Noire
- http://www.twin-pixels.com/the-making-of-avatar-some-juicy-details/
- http://www.twin-pixels.com/software-used-making-of-avatar/

Reflex behavior:

- ✓ A direct link between perception and actions (following behavior rules ("cells");
- ✓ Does not involve emotions.



Object behavior



Another example of reflex behavior – "Dexter" at MIT [Johnson, 1991]: Hand shake, followed by head turn

Agent behavior - avatars

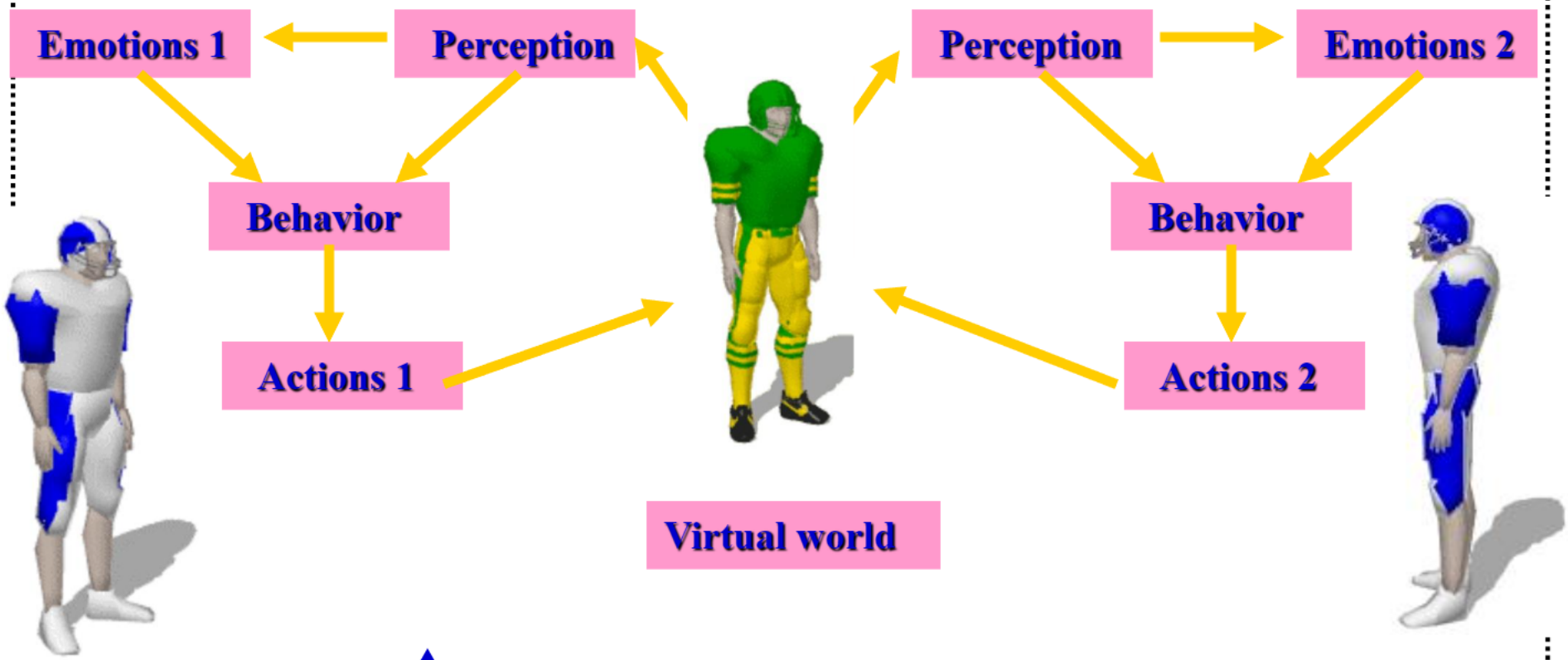
User-controlled body avatar

Autonomous virtual human

If user maps to a full-body avatar, then virtual human agents react through ***body expression recognition***: example dance. Swiss Institute of Technology, 1999 (credit Daniel Thalmann)

Emotional behavior:

- ✓ A subjective strong feeling (anger, fear) following perception;
- ✓ Two different agents can have different emotions to the same perception, thus they can have different actions.



Crowds behavior

- ✓ Crowd behavior emphasizes group (rather than individual) actions;
- ✓ Crowds can have guided LOA, when their behavior is defined explicitly by the user;
- ✓ Or they can have Autonomous LOA with behaviors specified by rules and other complex methods (including memory).

Political demonstration

Guided crowd
User needs to specify Intermediate path points

VC 5.3

Autonomous crowd
Group perceives info on its environment and decides a path to follow to reach the goal

(Thalmann et al., 2000)

MODEL MANAGEMENT

✓ It is necessary to maintain interactivity and constant frame rates when rendering complex models. Several techniques exist:

- Level of detail segmentation;
- Cell segmentation;
- Off-line computations;
- Lighting and bump mapping at rendering stage;
- Portals.

Level of detail segmentation:

- ✓ Level of detail (LOD) relates to the number of polygons on the object's surface. Even if the object has high complexity, its detail may not be visible if the object is too far from the virtual camera (observer).

Tree with 27,000 polygons

Tree with 27,000 polygons
(details are not perceived)

Static level of detail management:

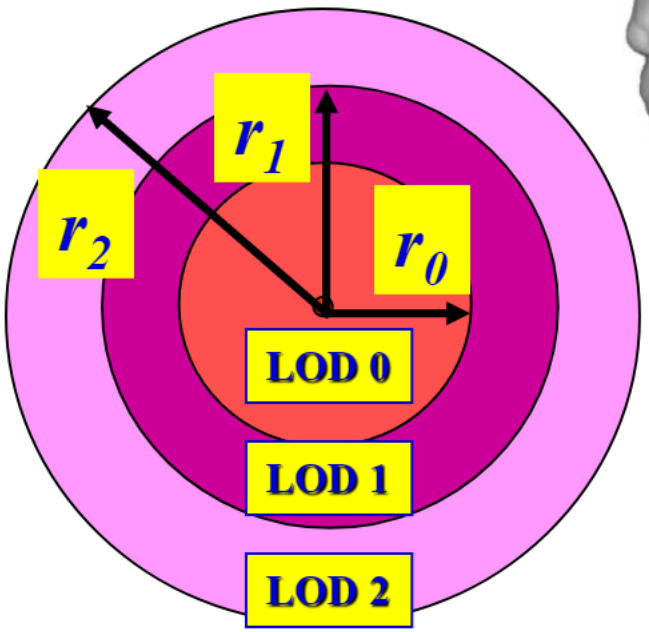
✓ Then we should use a simplified version of the object (fewer polygons), when it is far from the camera.

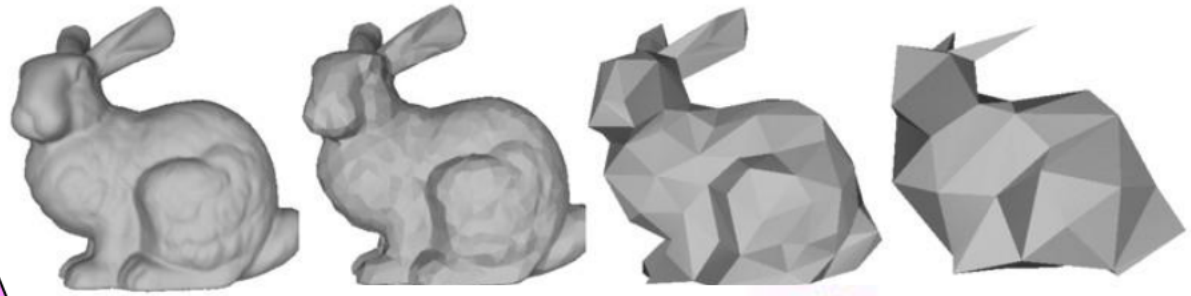
✓ There are several approaches:

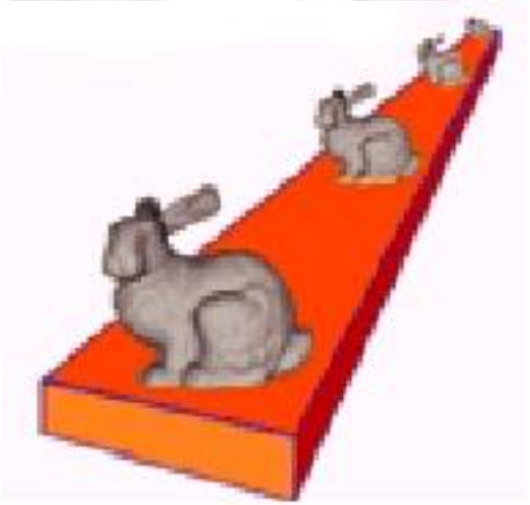
- Discrete geometry LOD;
- Alpha LOD;
- Geometric morphing (“geo-morph”) LOD.

Discrete Geometry LOD:

- ✓ Uses several discrete models of the same virtual object;
- ✓ Models are switched based on their distance from the camera $(r < r_0;\ r_0 < r < r_1;\ r_1 < r < r_2;\ r_2 < r)$



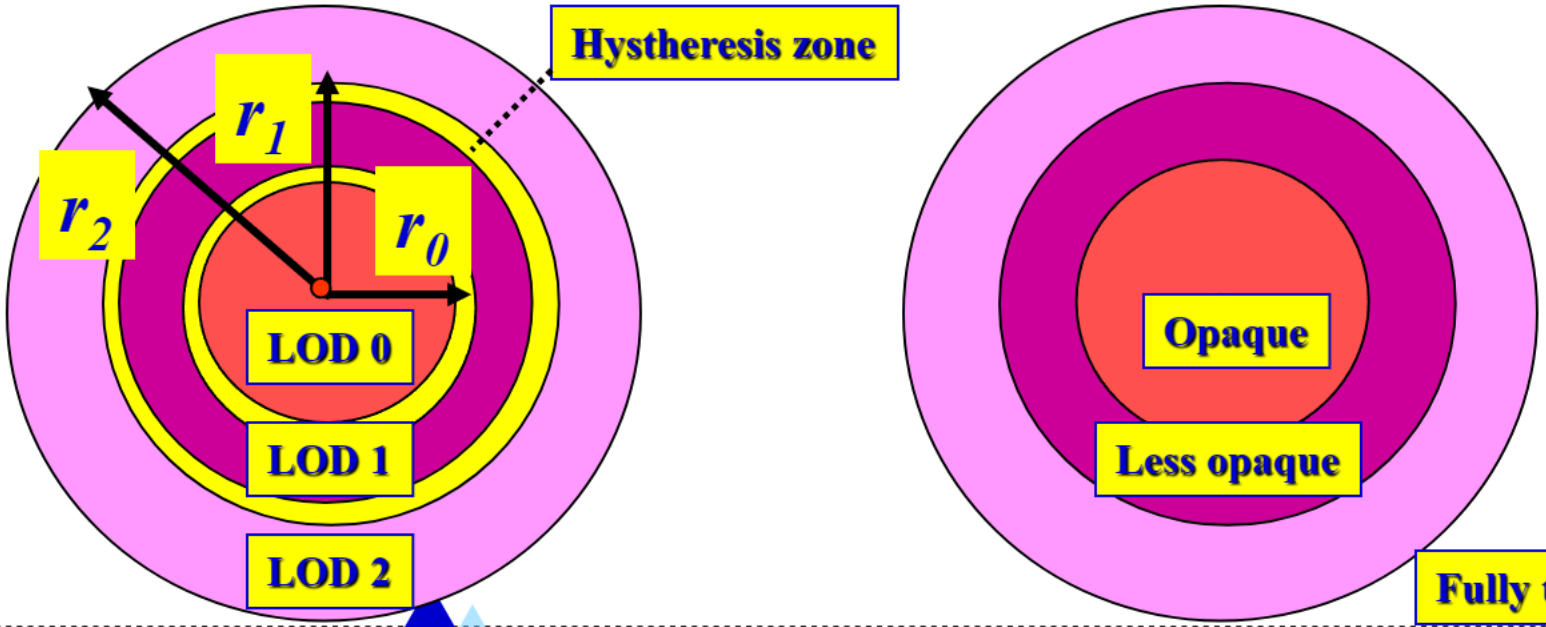




Alpha Blending LOD:

✓ Discrete LOD have problems on the $r_0 = r$, $r_1 = r$, $r_2 = r$ circles, leading to "popping". Objects appear and disappear suddenly. One solution is distance hystheresis. Another solution is model blending – two models are rendered near the circles;

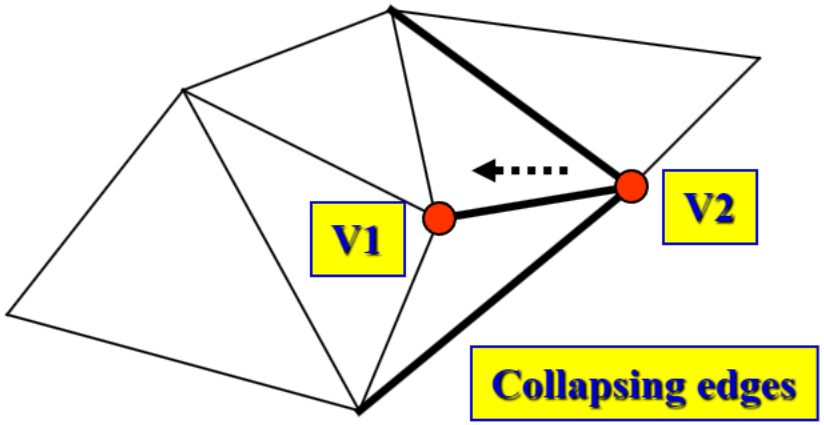
✓ Another solution to popping is ***alpha blending*** by changing the transparency of the object. Fully transparent objects are not rendered.



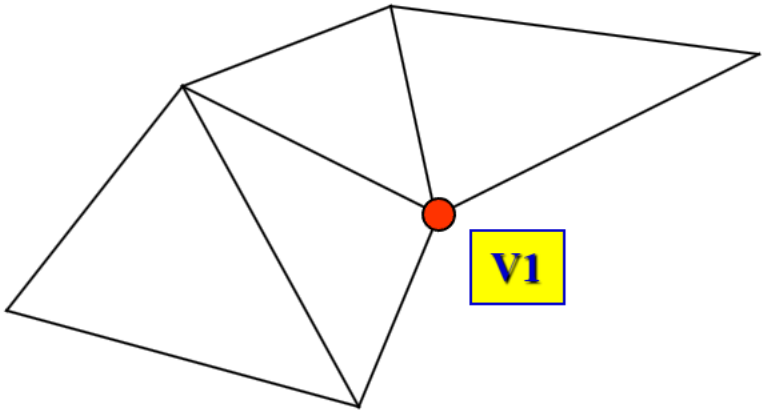
Geometric Morphing LOD:

- ✓ Unlike geometric LOD, which uses several models of the same object, geometric morphing uses only one complex model.
- ✓ Various LOD are obtained from the base model through ***mesh simplification***
- ✓ A triangulated polygon mesh: ***n*** vertices has ***2n*** faces and ***3n*** edges

Mesh *before* simplification



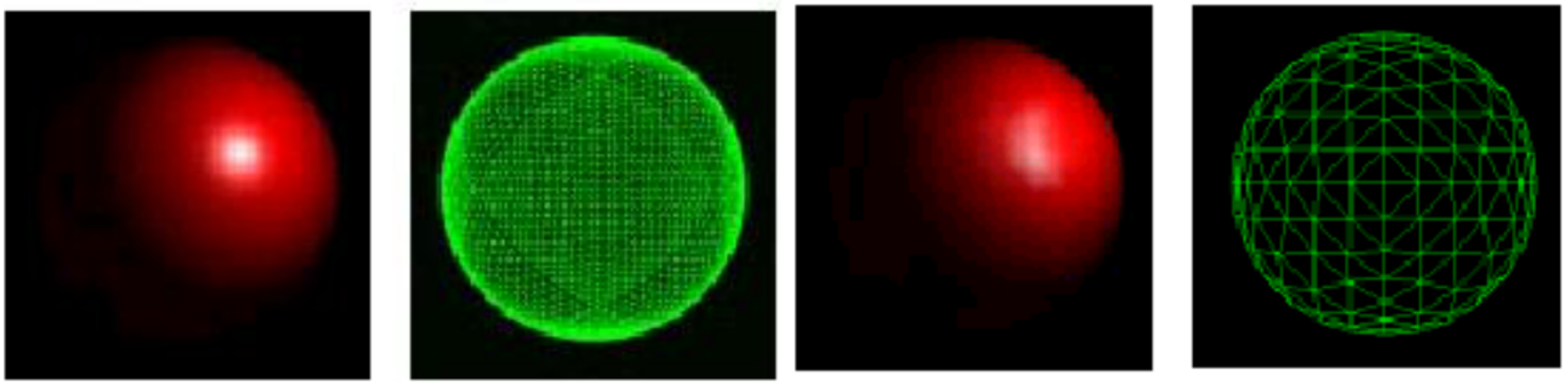
Mesh *after* simplification



Single-Object adaptive level of detail LOD:

✓ Used where there is a single highly complex object that the user wants to inspect (such as in interactive scientific visualization.

✓ Static LOD will not work since detail is lost where needed- example the sphere on the right loses shadow sharpness after LOD simplification.



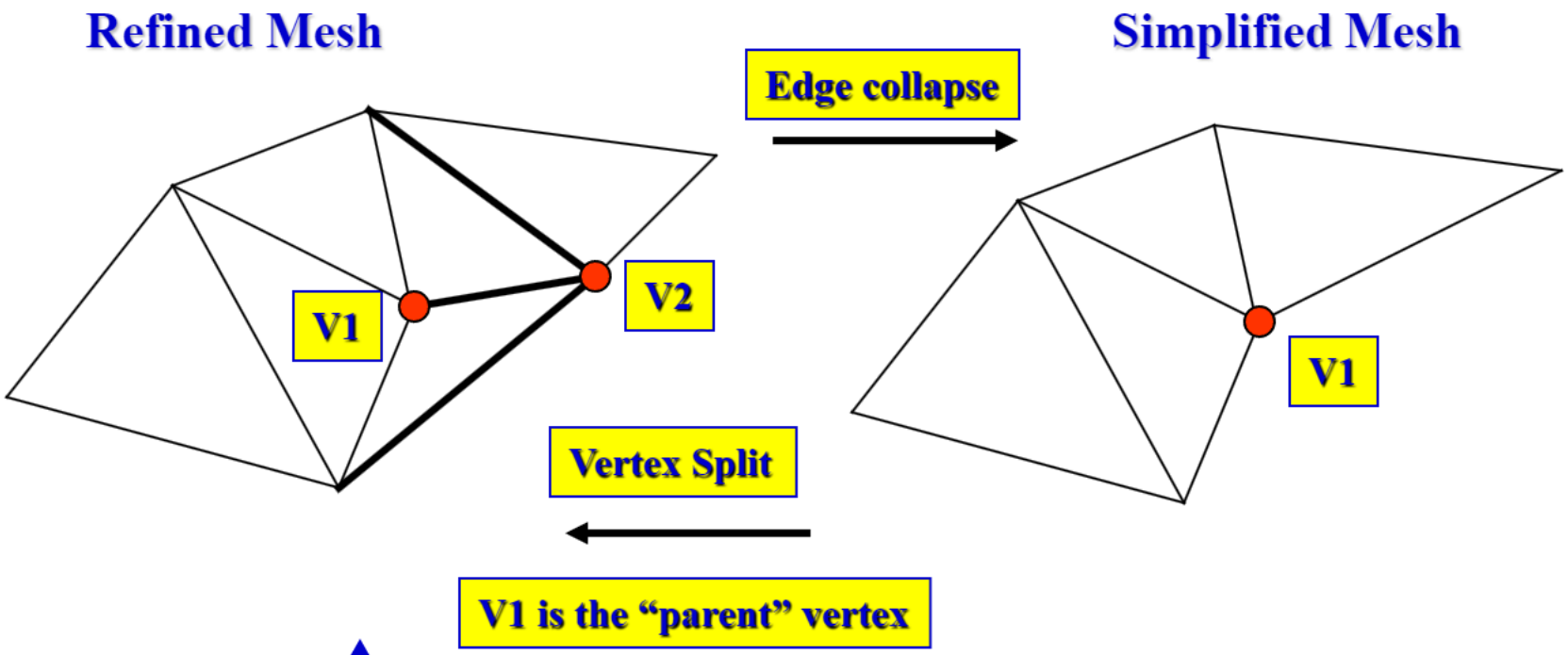
Sphere with 8192 triangles – Uniform high density

Sphere with 512 triangles – Static LOD simplification

(from Xia et al, 1997)

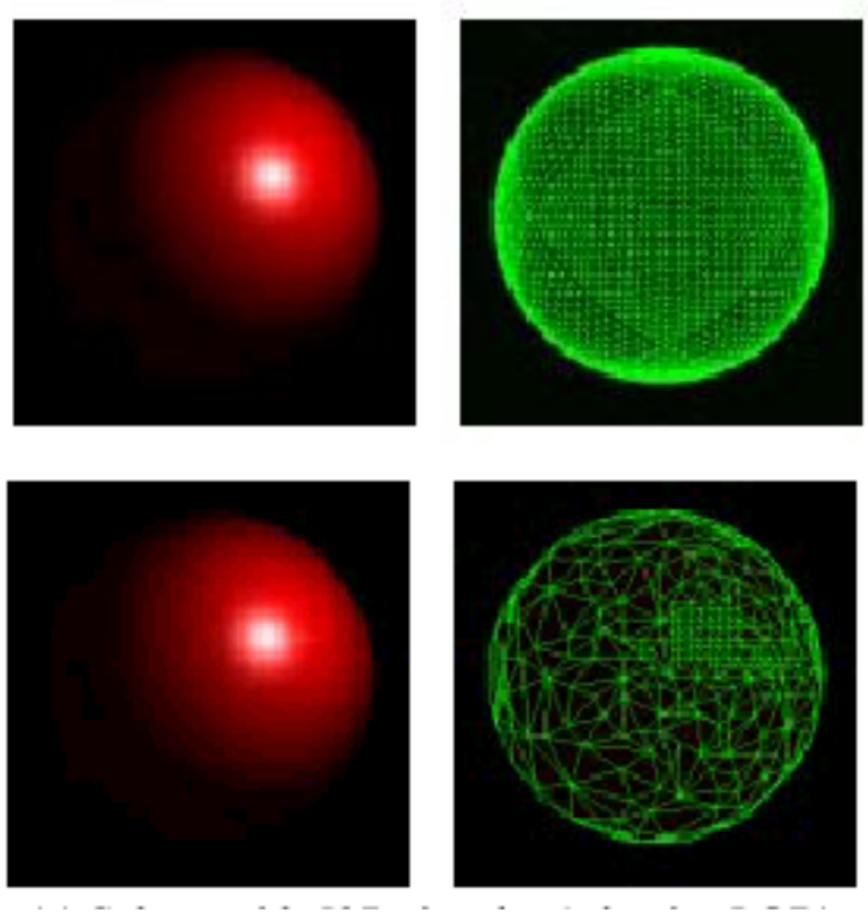
Single-object Adaptive Level of Detail

Sometimes edge collapse leads to problems, so vertices need to be split again to regain detail where needed. Xia et al. (1997) developed an adaptive algorithm that determines the level of detail based on distance to viewer as well as normal direction (lighting).



(adapted from Xia et al, 1997)

Single-object Adaptive Level of Detail

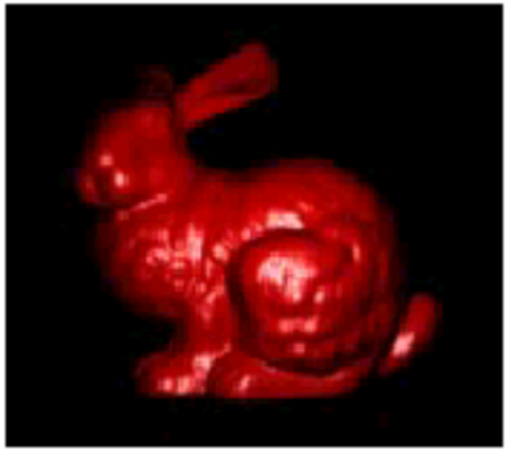


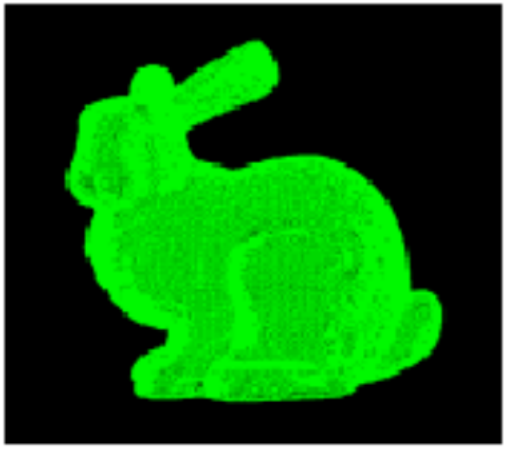
Sphere with 8192 triangles – Uniform high density, 0.115 sec to render

Sphere with 537 triangles – adaptive LOD, 0.024 sec to render (SGI RE2, single R10000 workstation)

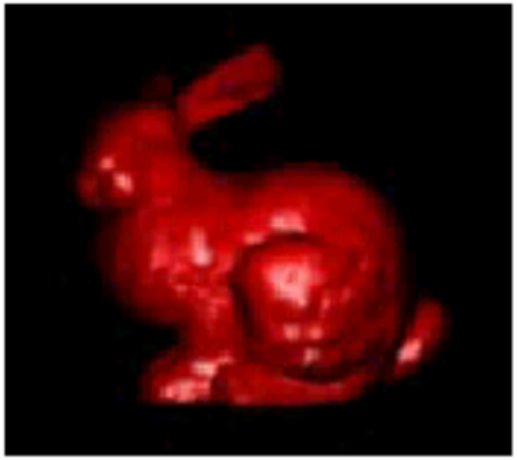
(from Xia et al, 1997)

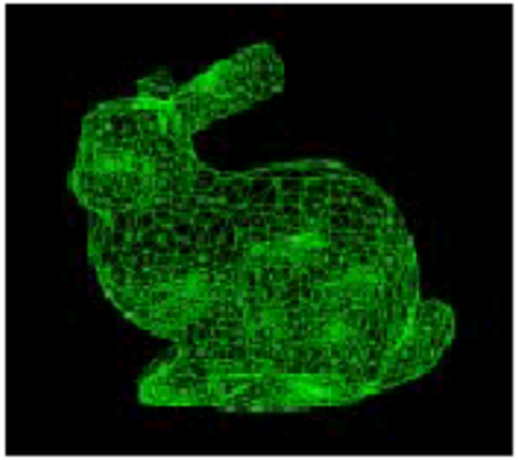
Single-object Adaptive Level of Detail





Bunny with 69,451 triangles – Uniform high density, 0.420 sec to render



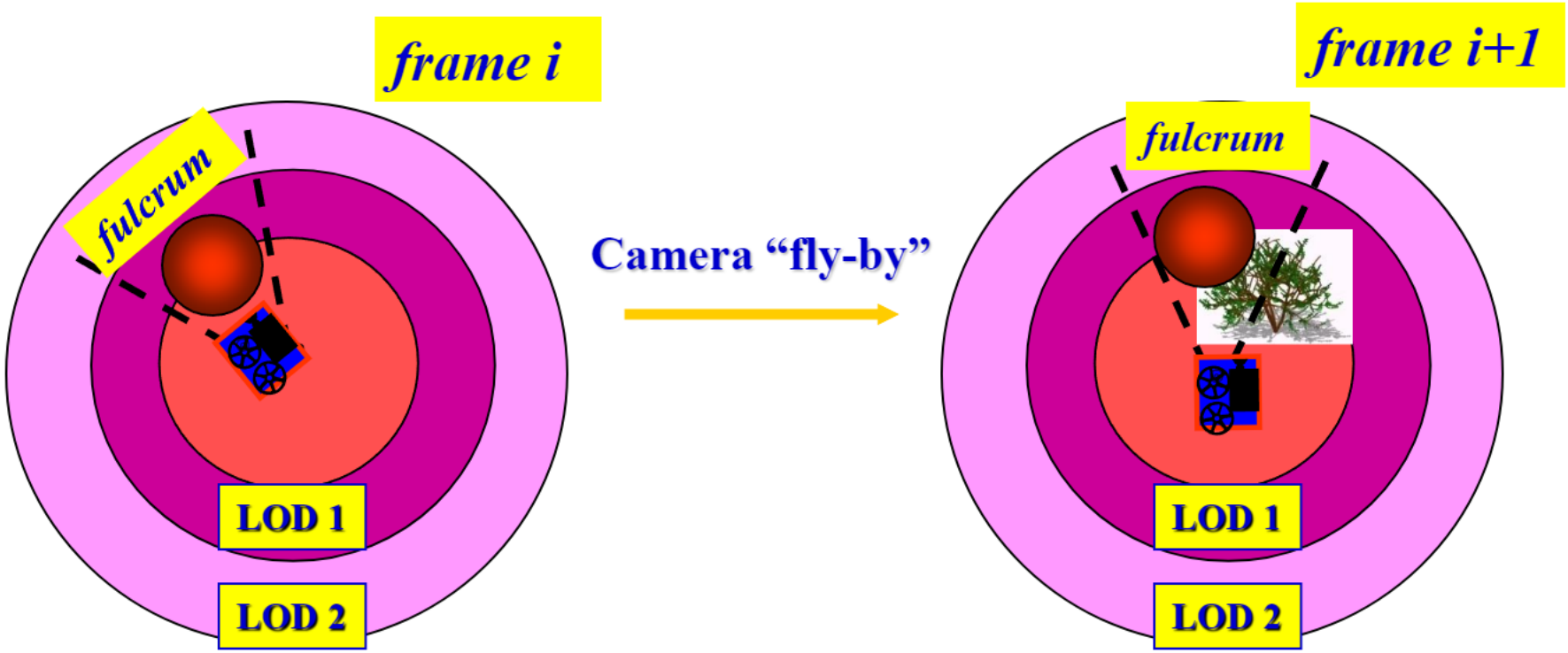


Bunny with 3615 triangles – adaptive LOD, 0.110 sec to render (SGI RE2, single R10000 workstation)

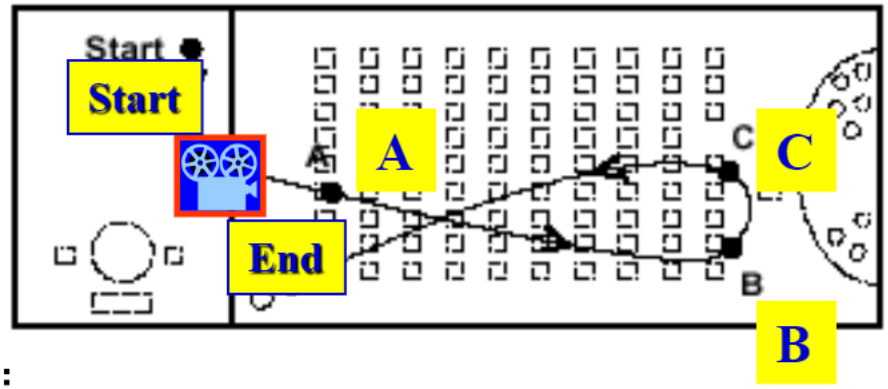
(from Xia et al, 1997)

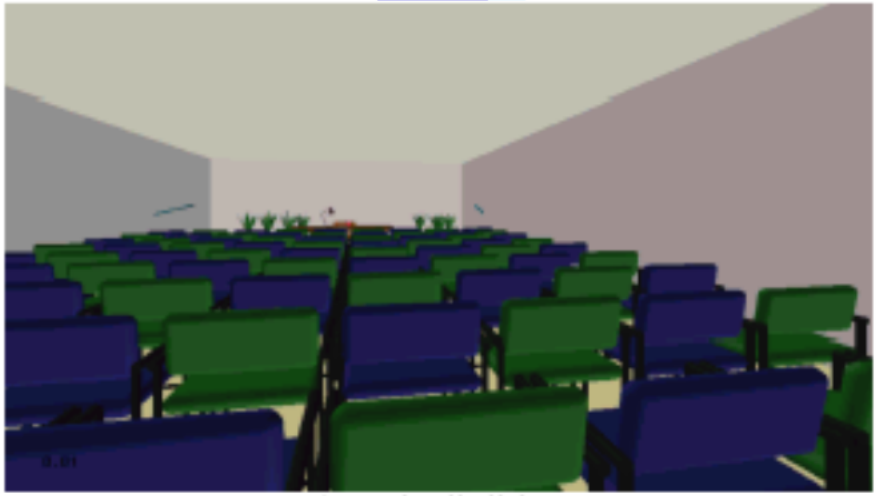
Static LOD:

✓ Geometric LOD, alpha blending and morphing have problems maintaining a *constant* frame rate. This happens when new complex objects appear *suddenly* in the scene (fulcrum).



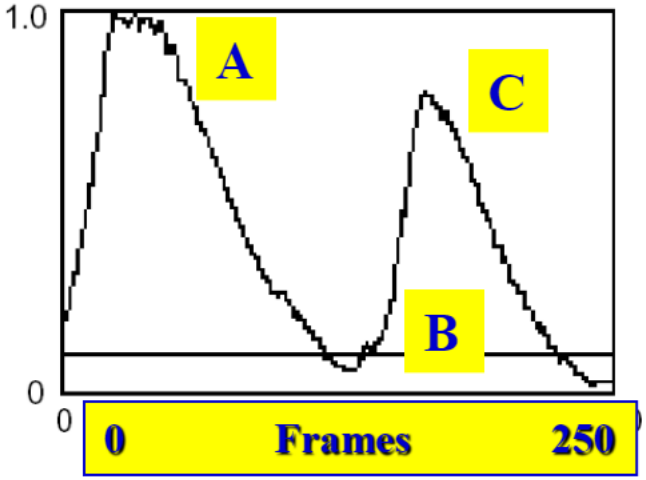
Architectural "walk-through" (UC Berkeley Soda Hall)





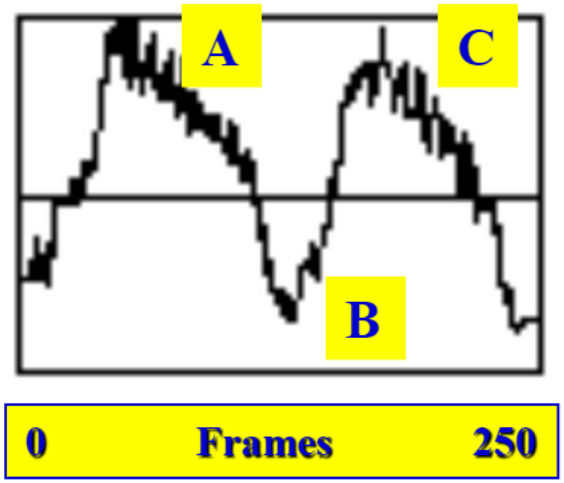
No LOD menegament, 72,570 polygons

Time 1.0 sec



No LOD management

Time 0.2 sec



Static LOD management

from (Funkhauser and Sequin, 1993)

Adaptive LOD Management-continued:

✓ An algorithm that selects LOD of visible objects based on a specified frame rate;

✓ The algorithm (Funkhauser and Sequin, 1993) is based on a benefits to cost analysis, where cost is the time needed to render Object ***O*** at level of detail ***L,*** and rendering mode ***R.***

✓ The cost for the whole scene is

$$\Sigma \text{ Cost } (O,L,R) \leq \text{Target frame time}$$

✓ where the cost for a single object is

$$\text{Cost } (O,L,R) = \max (c1\,\text{Polygons}(O,L) + c2\,\text{Vertices}(O,L),\ c3\,\text{Pixels}(O,L))$$

c1, c2, c3 are experimental constants, depending on R and type of computer

Adaptive LOD Management:

✓ Similarly the benefit for a scene is a sum of visible objects benefits;

$$\Sigma \text{ Benefit}(O,L,R)$$

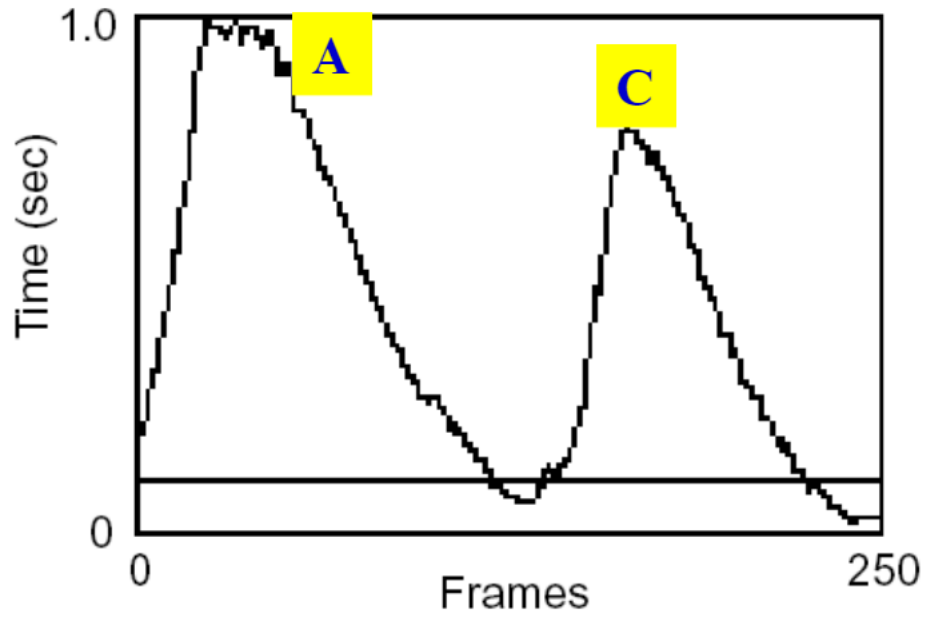
✓ where the benefit of a given object is

$$\text{Benefit}(O,L,R) = \text{size}(O) * \text{Accuracy}(O,L,R) * \text{Importance}(O) * \text{Focus}(O) * \text{Motion}(O) * \text{Hysteresis}(O,L,R)$$

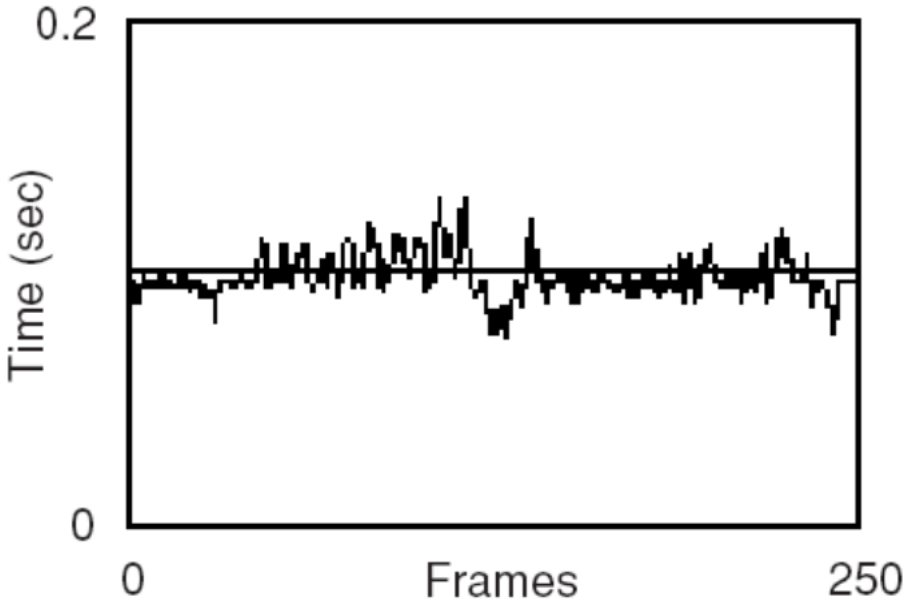
✓ The algorithm tries to maximize each object's "value"

$$\text{Value} = \text{Benefit}(O,L,R)/\text{Cost}(O,L,R)$$

✓Objects with higher value (larger size) are rendered first



No detail elision, 72,570 polygons



Optimization algorithm, 5,300 poly.
0.1 sec target frame time (10 fps)

from (Funkhauser and Sequin, 1993)

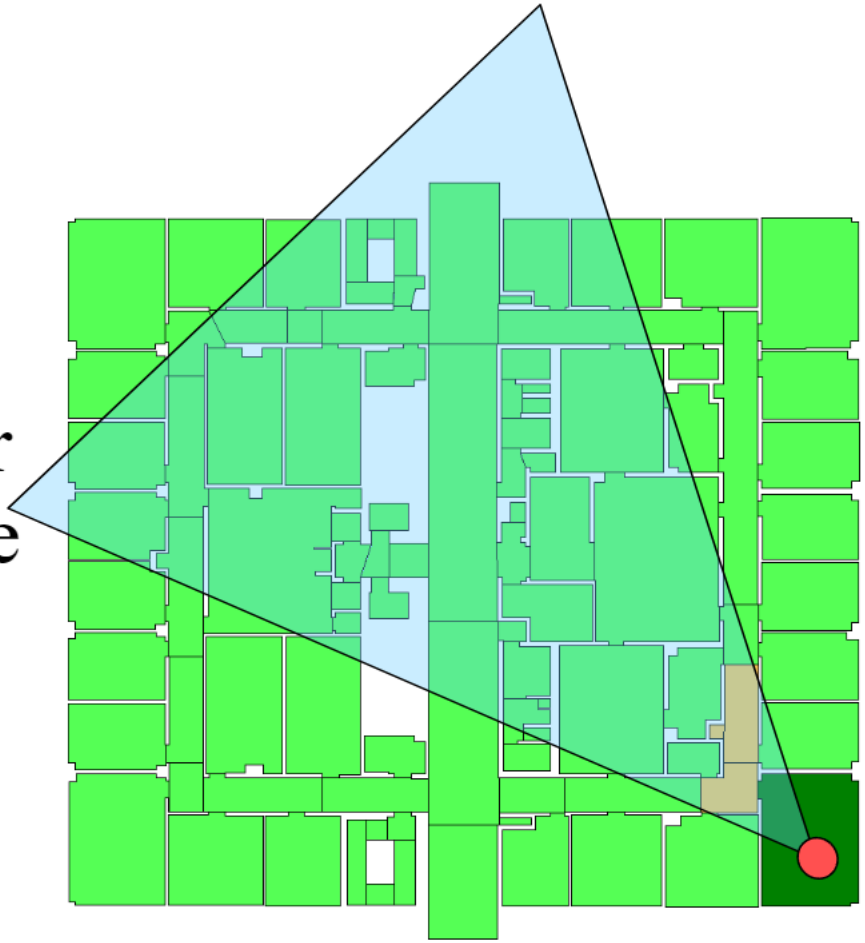
Cell segmentation:

- ✓ It is another method of model management, used in architectural walk-through;
- ✓ To maintain the "virtual building" illusion it is necessary to have at least 6 fps (Airey et al., 1990)
- ✓ Necessary to maintain ***interactivity*** and ***constant frame rates*** when rendering complex models.

The Visibility Problem

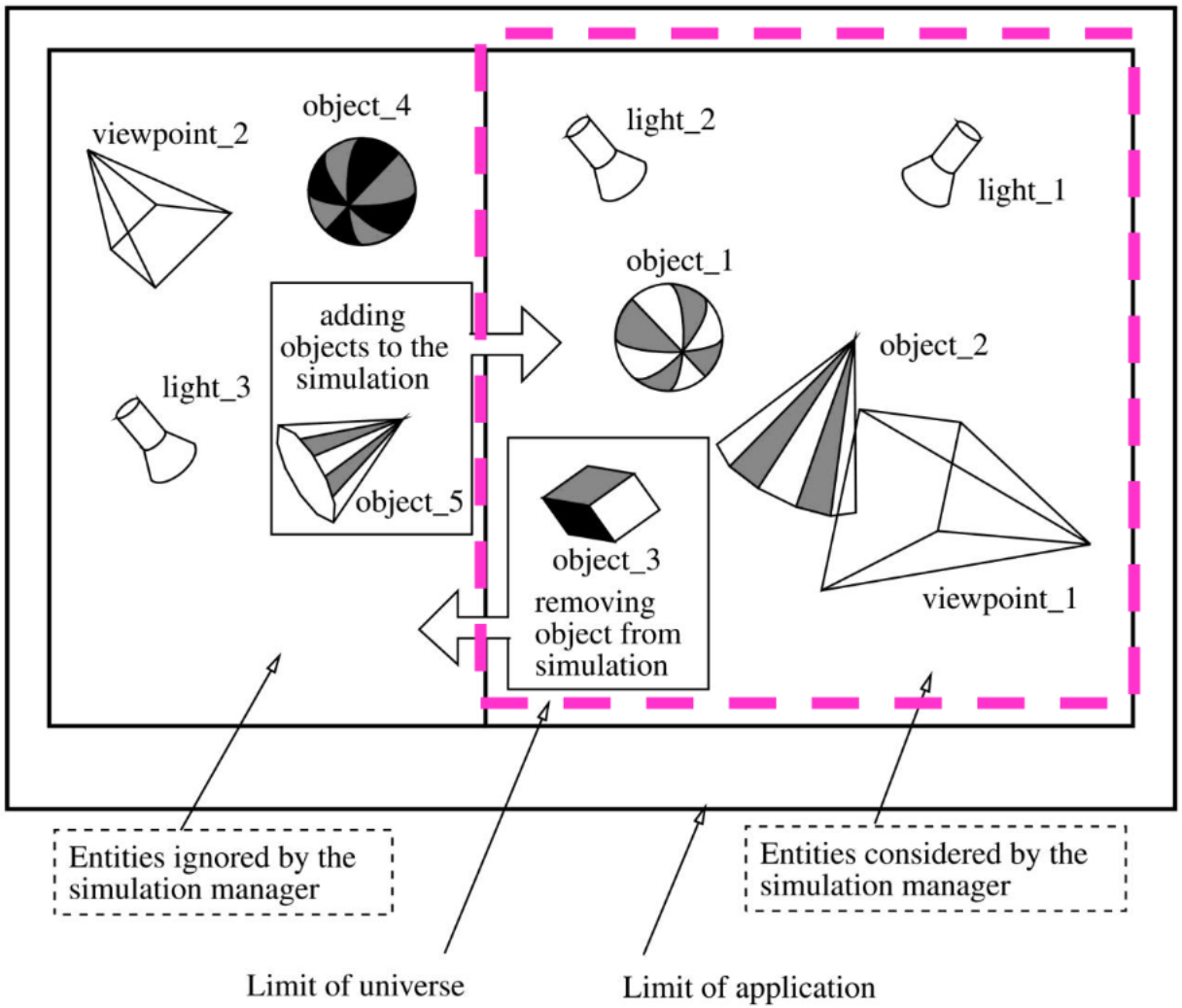
Given the current view point and the viewing direction.

We would like to render the visible portion of the model and as little as possible of the rest of the model.



Model management

Only the current "universe" needs to be rendered



Cells and Portals

Cells:

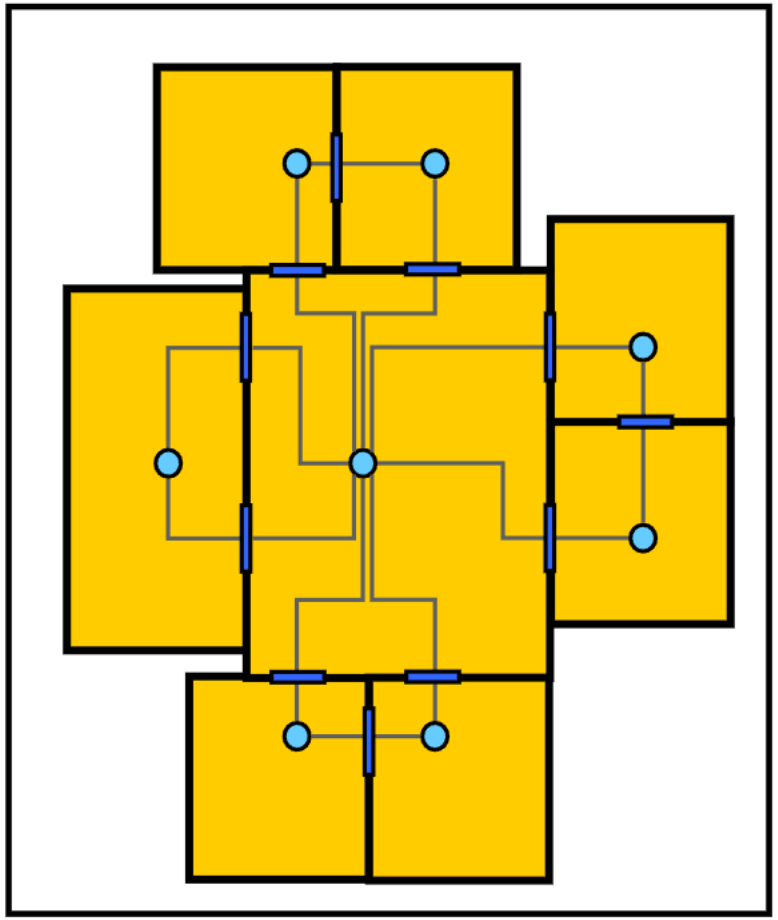
Regions whose boundary is formed by a sequence of walls and portals.

For example: rooms.

Portals:

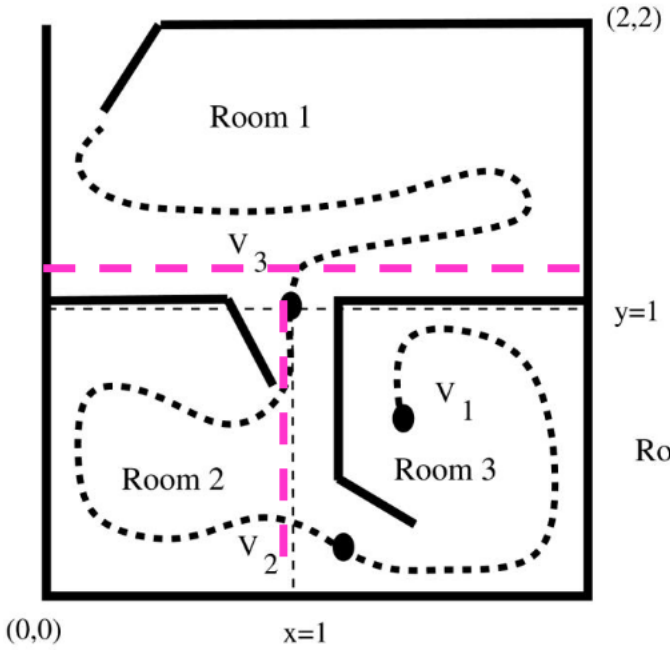
Transparent doorways connecting two cells, filling the missing portions of the cells boundary.

For example: doors.

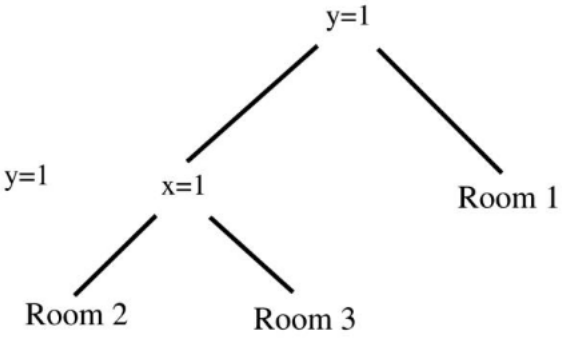


Cell segmentation – increased frame rate

- ✓ Buildings are large models that can be partitioned in "cells" automatically and off-line to speed up simulations at run time;
- ✓ Cells approximate rooms;
- ✓ Partitioning algorithms use a "priority" factor that favors occlusions (partitioning along walls)



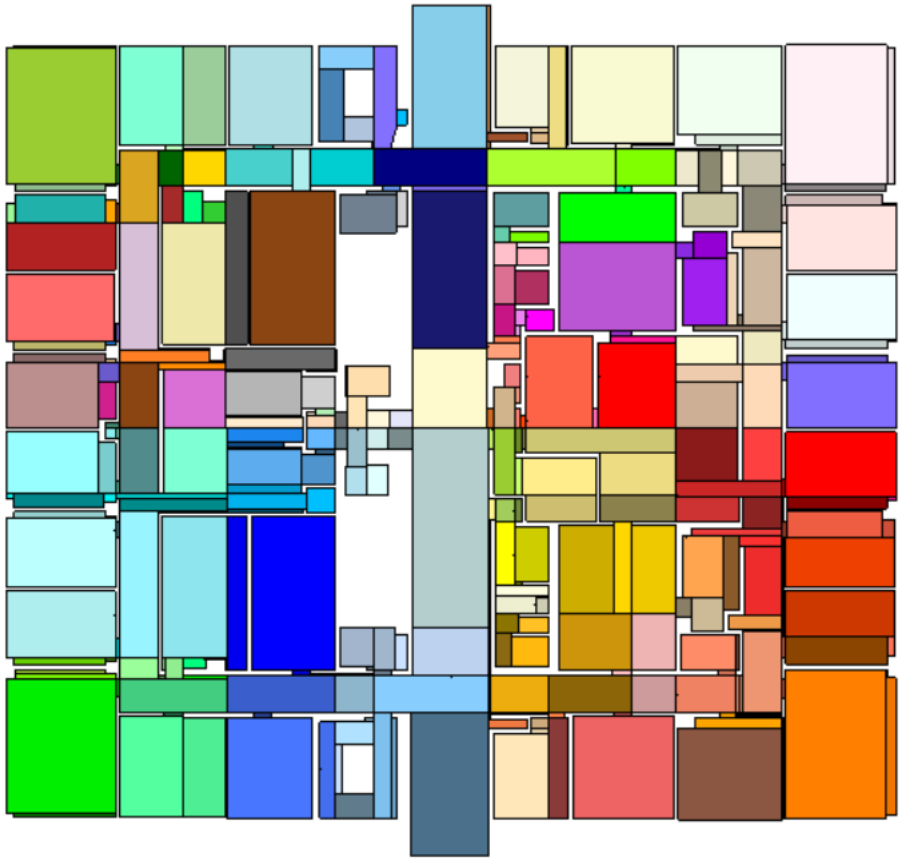
Binary Space Partition (BSP) tree



a) Automatic floor plan partition (Airey et al., 1990)

BSP Trees Algorithm

- Choose a splitting plane. Use the plane to split the space into two.
- Repeat recursively for both subspaces.
- The end result is a binary tree whose leafs are convex spaces, which are the cells.



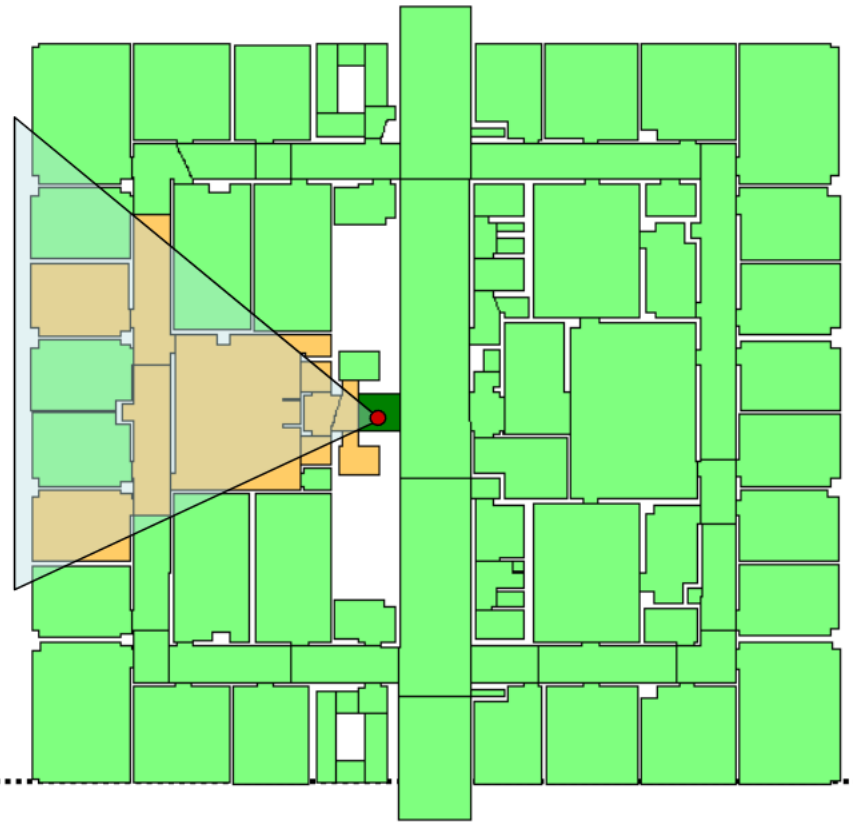
Rendering the cells

Cells and Portals work under the assumption that the model is hidden and that the visible parts should be discovered.

The current cell

Visible cells

Hidden cells



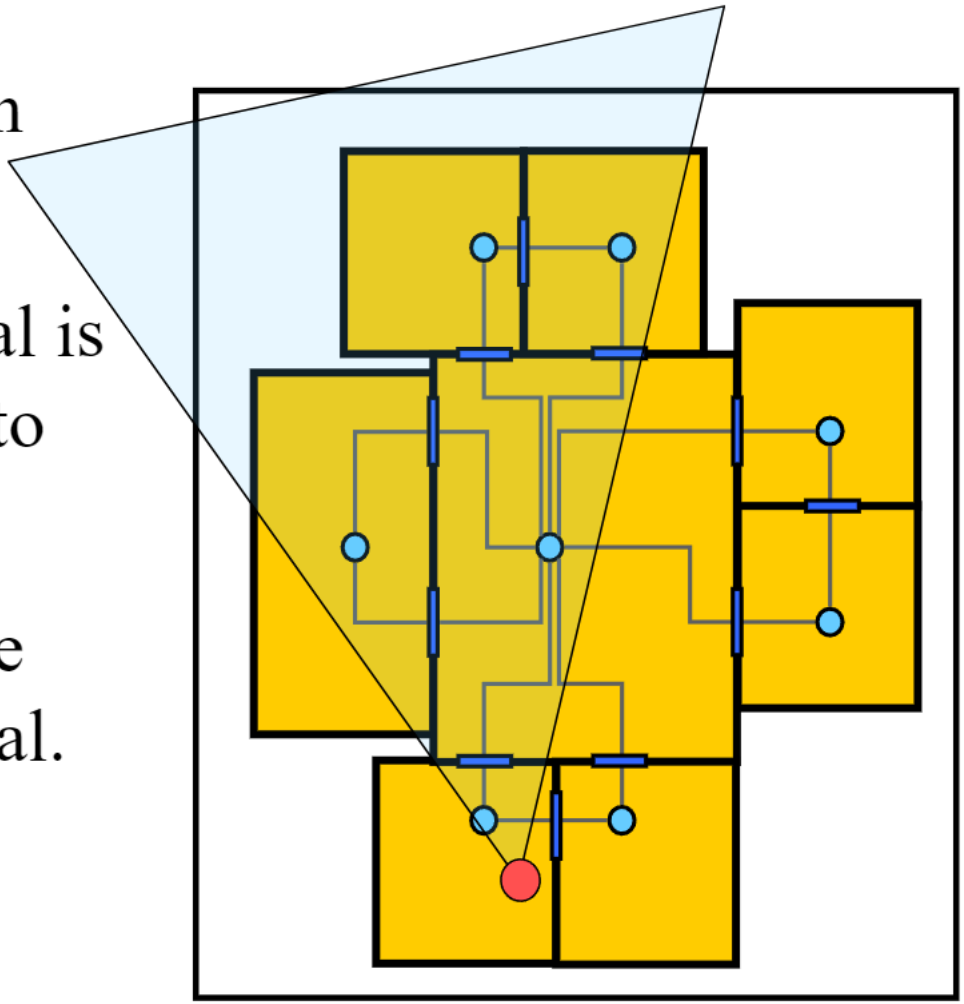
Usually, visibility algorithms assume the model is visible and that the hidden parts should be removed.

Rendering the cells

Run a Depth First Search from the current cell.

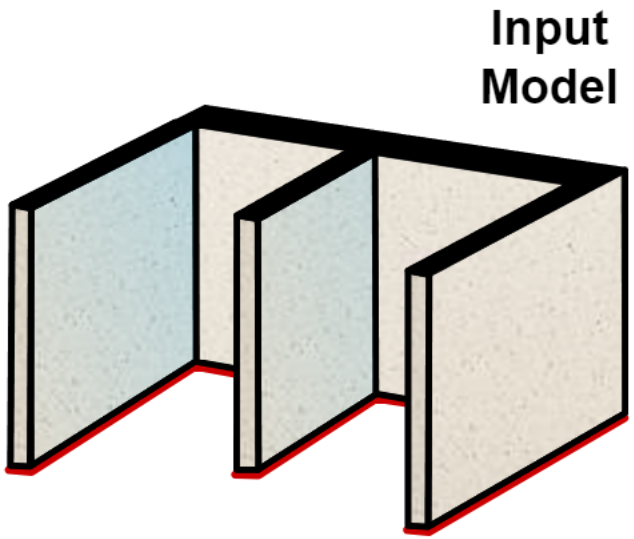
Test the portals. If a portal is visible, clip it and move to the adjacent cell.

Render the cells that were visited during the traversal.



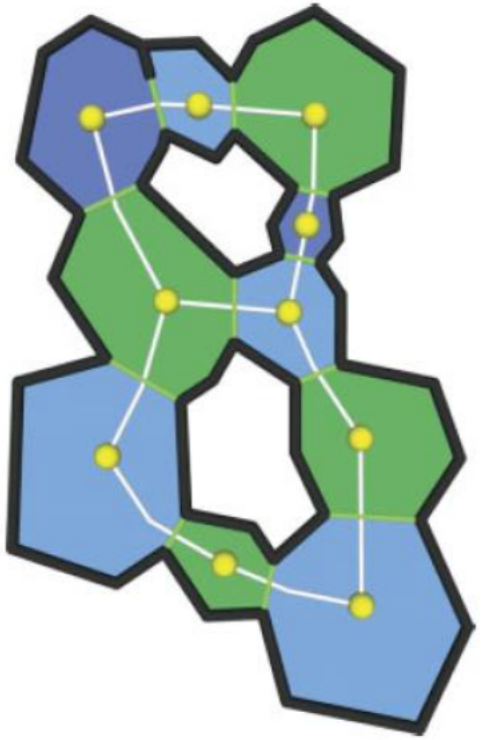
Breaking the Wall Algorithm

- The algorithm works in 2D.
- The edges given to the algorithm are extracted from the 3D model. They are single sided and oriented.
- Similarly, the created cells can be given height and associated with 3D model.



Breaking the Wall Algorithm

- Uses a adjacency graph-
- Cells are nodes and portals are branches
- The set of Cells that are visible from the current view point is found through a recursive depth-first traversal of the adjacency graph.
- When applying the algorithm only the geometry from the partially visible cells is sent to the graphics pipeline



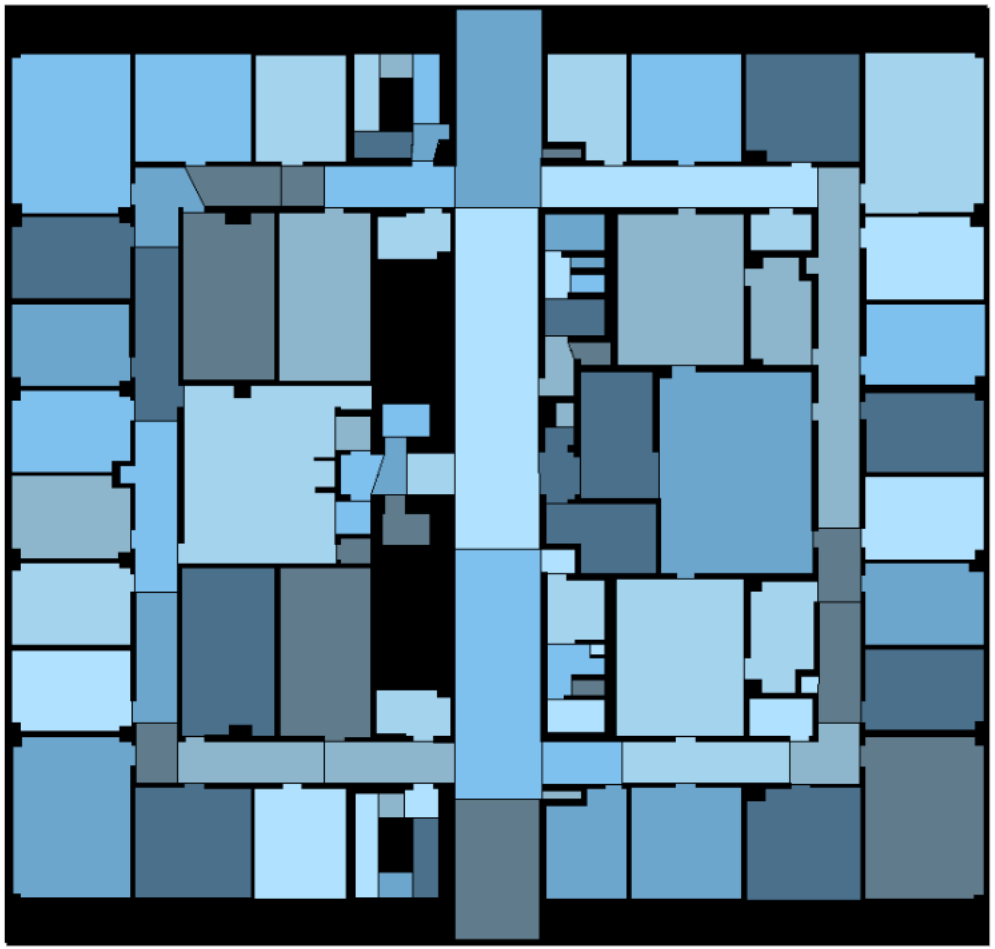
Breaking the Wall Algorithm

Initial partition:

Create cells with short portals relative to wall size.

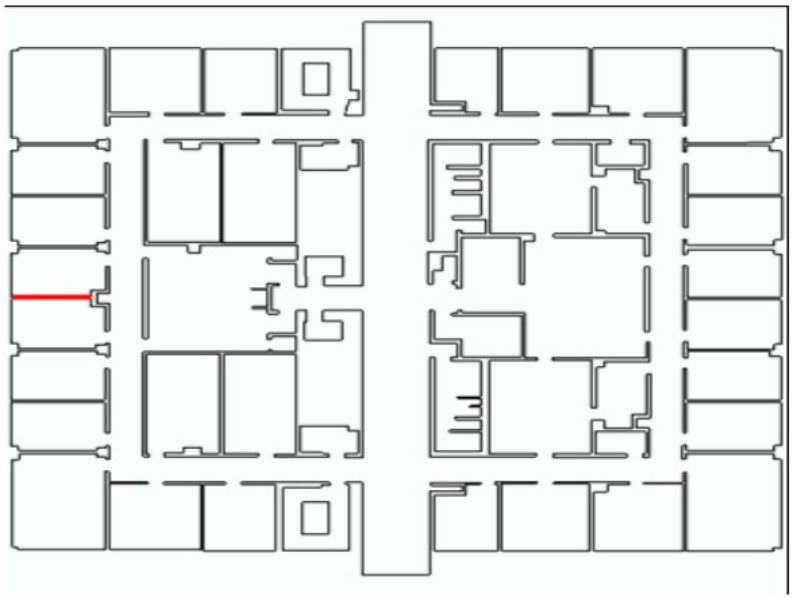
Final partition:

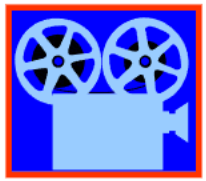
Refine the partition, creating smaller cells that have short portals relative to cell size.



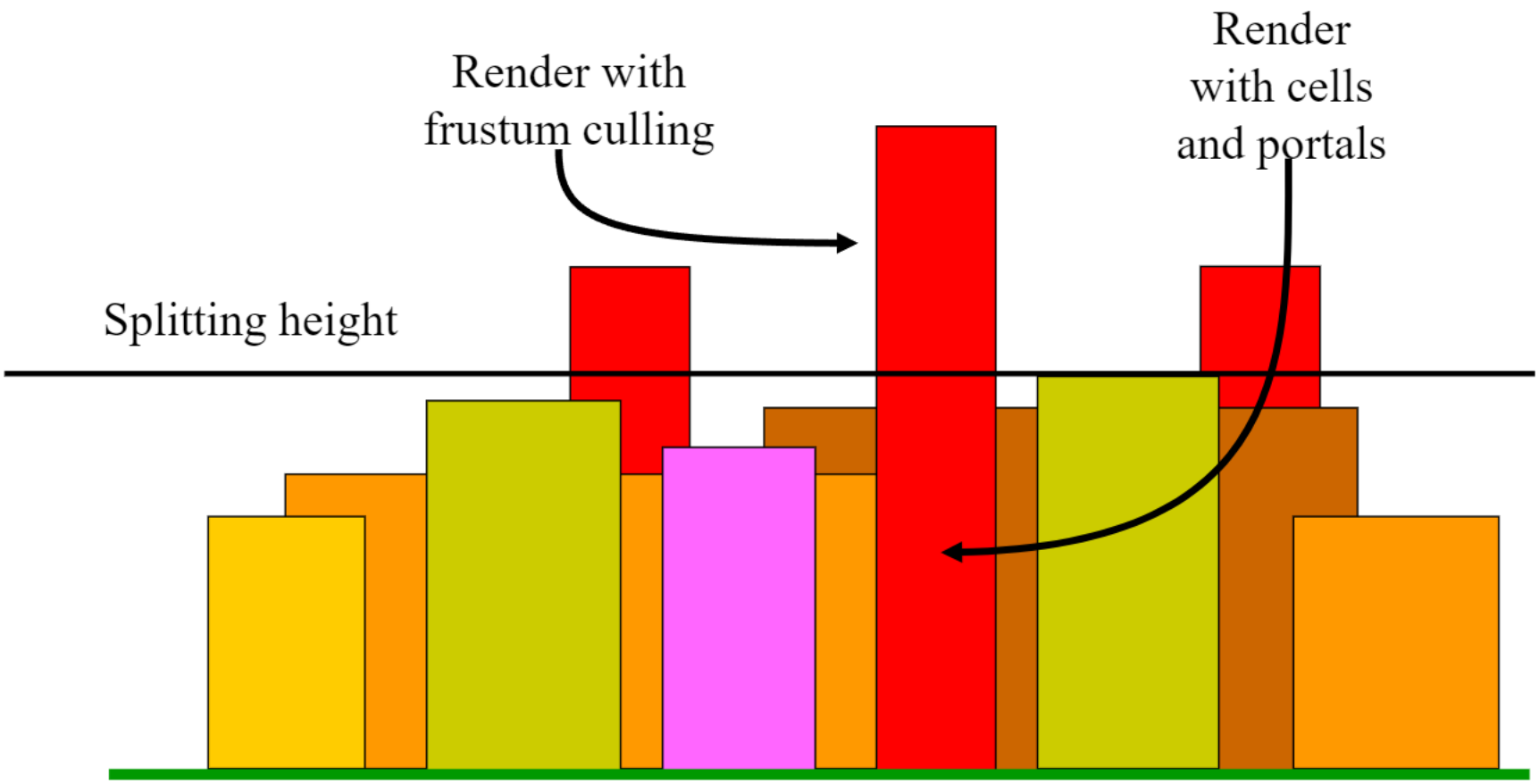
Creating the initial Partition

- Traverse the walls (ccw), creating portals wherever necessary.
- Whenever the path intersects itself, a cell is created. Remove the cell from the path and continue.
- Repeat the process until all the walls have been assigned to cells.



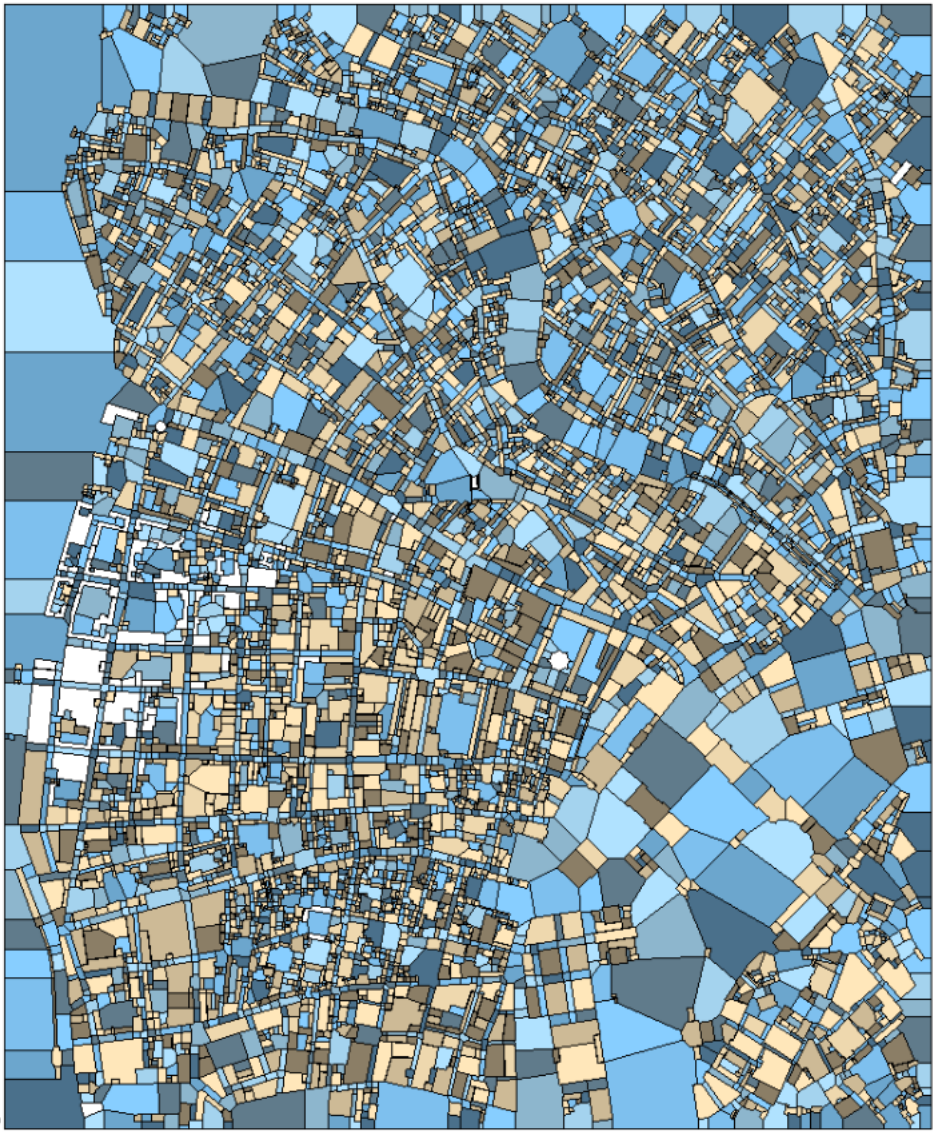


Going outdoors



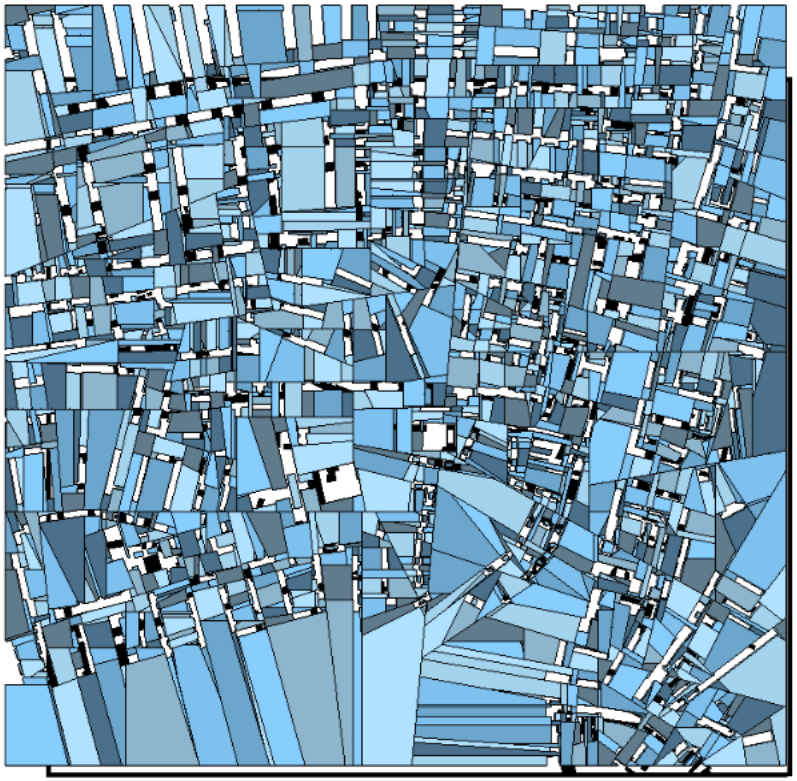
Going outdoors

- Create a **ground** partition.
- Create a **roof-top** partition.
- Merge the partitions and extend the portals.

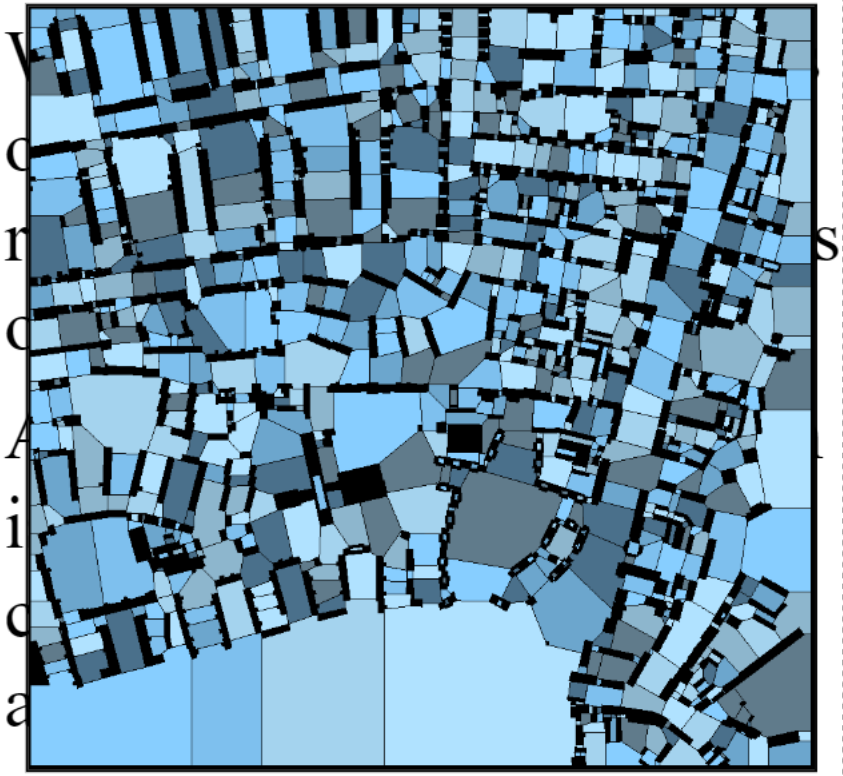


Evaluating the Partitions Quality

BSP Partition



BW Partition



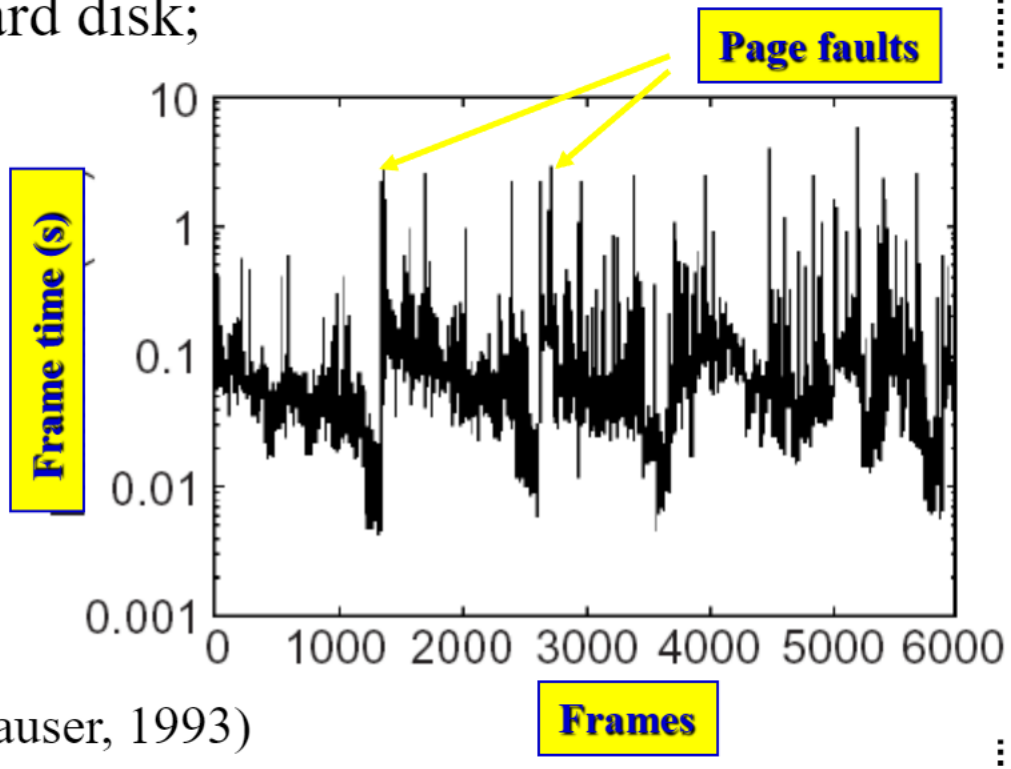
“Merged BSP” Partition

Evaluating the Partitions Quality

Model	Method	Avg. #visible cells	Avg. # portals tested	Avg. area rendered
Soda	BW	8	25	106.247
	"Merged BSP"	12	85	116.739
London	BW	25	83	72.551
	"Merged BSP"	28	609	107.986
Sava	BW	18	62	956.454
	"Merged BSP"	23	372	1959.15

Cell segmentation

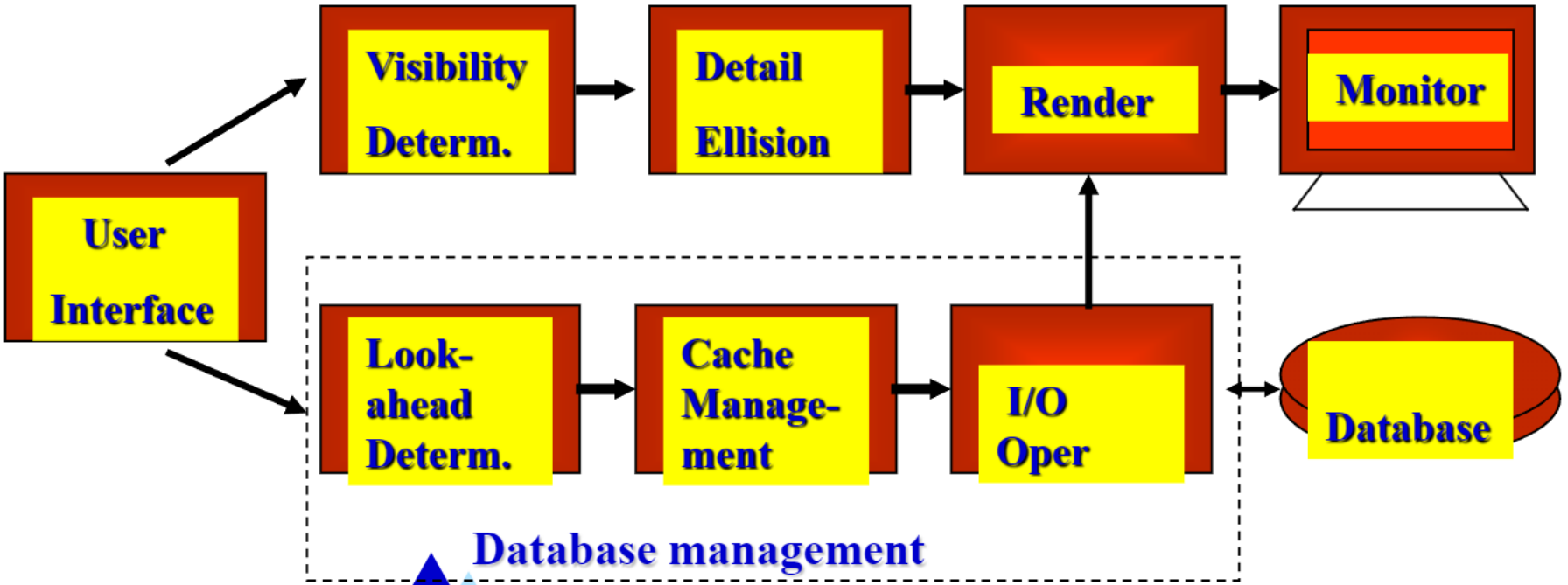
- ✓ Building model resides in a fully associative cache;
- ✓ But cell segmentation alone will not work if the model is so large that it exceeds available RAM;
- ✓ In this case large delays will occur when there is a page fault and data has to be retrieved from hard disk;



From (Funkhauser, 1993)

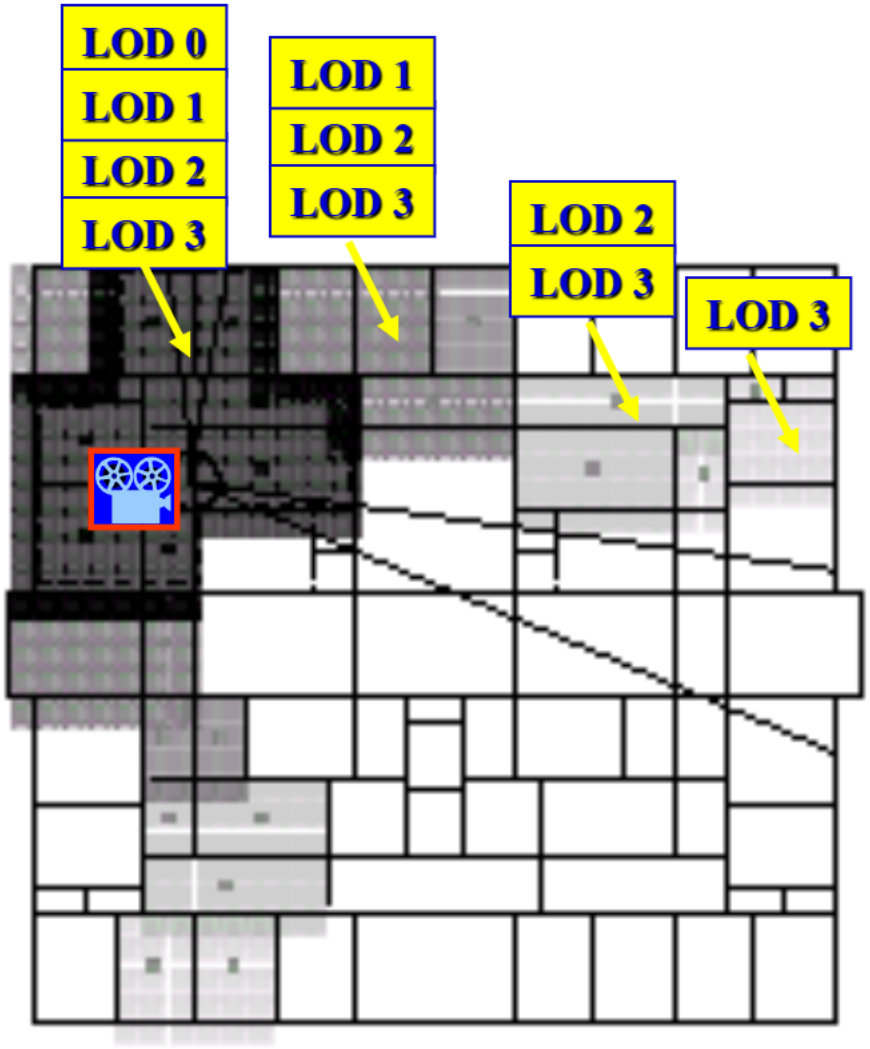
Combined Cell, LOD and database methods

- ✓ It is possible to add database management techniques to prevent page faults and improve fps uniformity during walk-through;
- ✓ It is possible to estimate how far the virtual camera will rotate and translate ***over the next N frames*** and pre-fetch from the hard disk the appropriate objects.

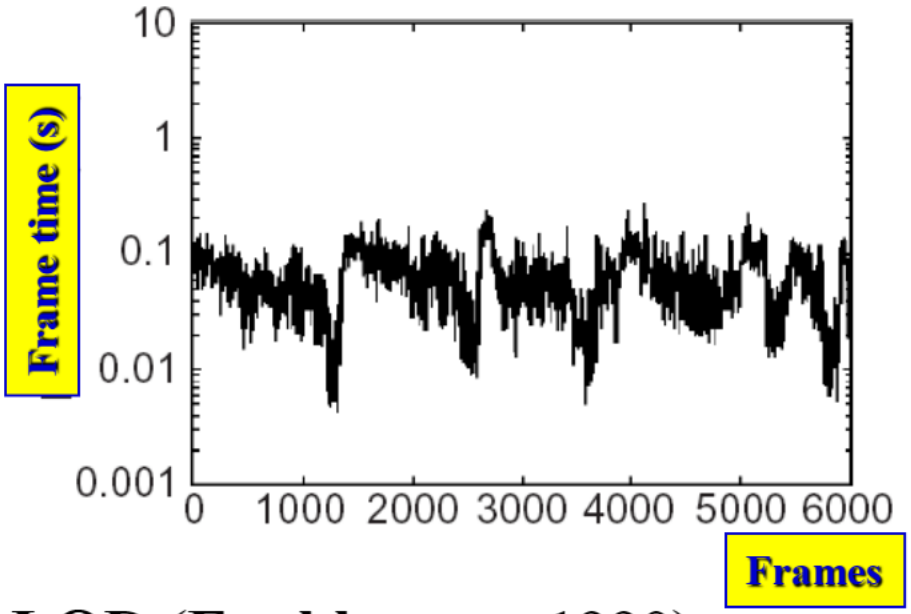


Floor plan partition (Funkhouser, 1993)

Database management



LOD 0 – highest level of detail (loaded first)
….
LOD 3 - lowest level of detail (loaded last)



Floor plan visibility and highest LOD (Funkhouser, 1990)