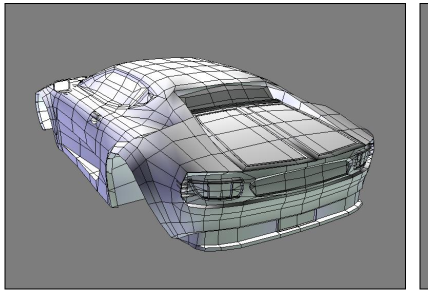
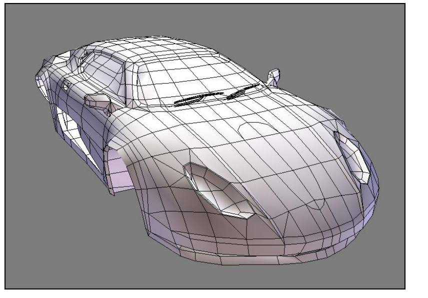
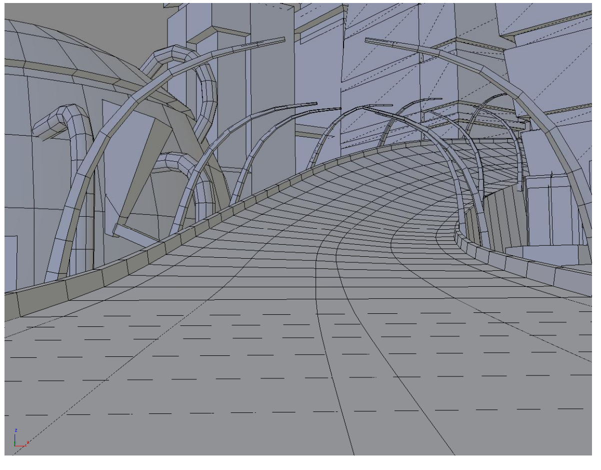
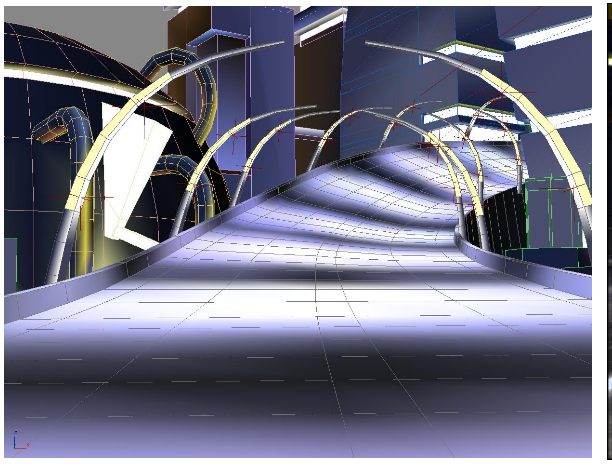
Design of Human Interface Game Software

– Modeling

Modeling for Games

- Modeling Surfaces
- Modeling Context
  - Scene creation
  - Modeling Illumination

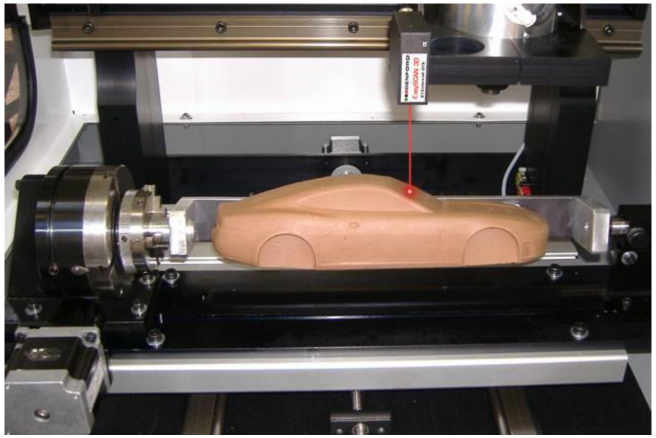



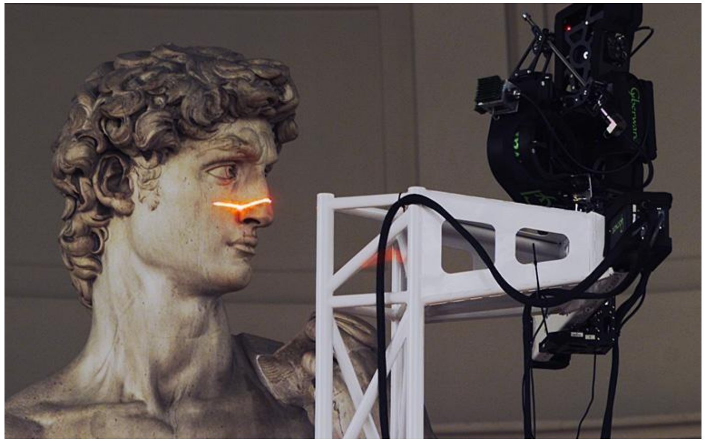




Images courtesy of WildTangent.

Model Creation

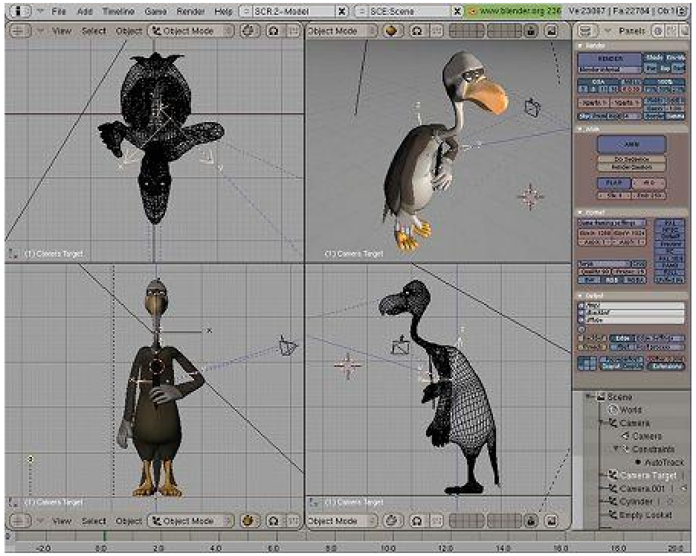
- Real world objects or clay models can be scanned or digitized
- May not save time because of complicated cleanup, but will ensure high fidelity

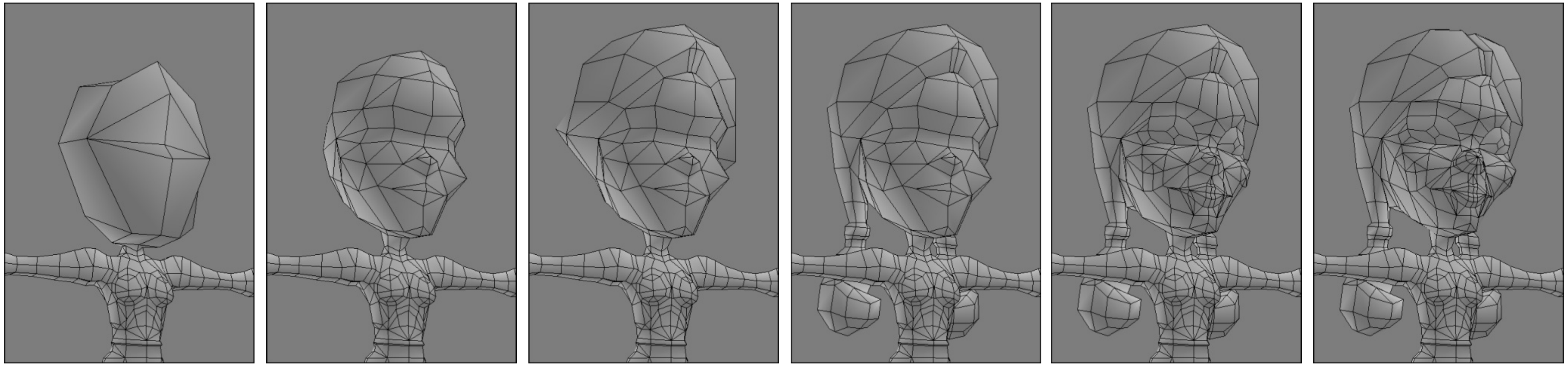




Model Creation

- Using modeling tools
  - 3DS Max
  - Maya
  - Blender3D (Open source)





CAU

3D Models on the web

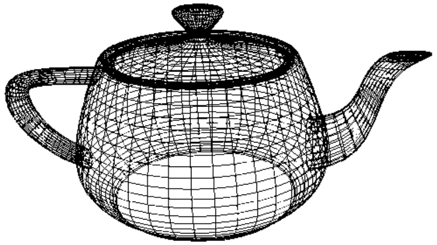
- Free Models
  - search on google!
    - http://opengameart.org/
    - http://free3d.com
    - http://www.turbosquid.com/
  - Modeling tool resources
    - Blender (http://www.blender-models.com/)
  - Game engine resources
    - Unity, Unreal, Ogre3D, …
    - 3D models, textures, sounds

Game Engines

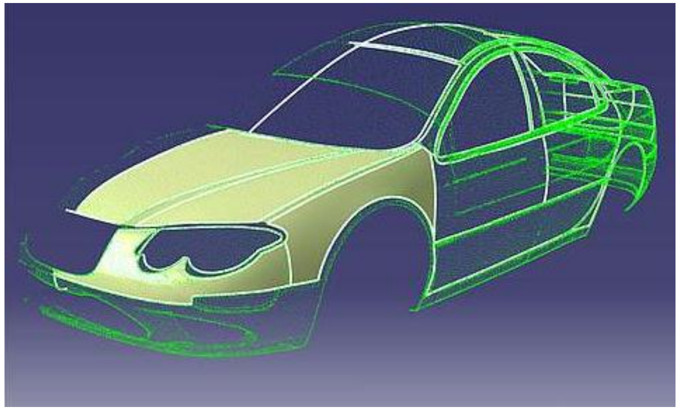
- Game Engines
  - Open Source
    - Ogre3D
    - Godot
    - Panda3D
  - Commercial
    - Unity
    - Unreal

Representing Surfaces

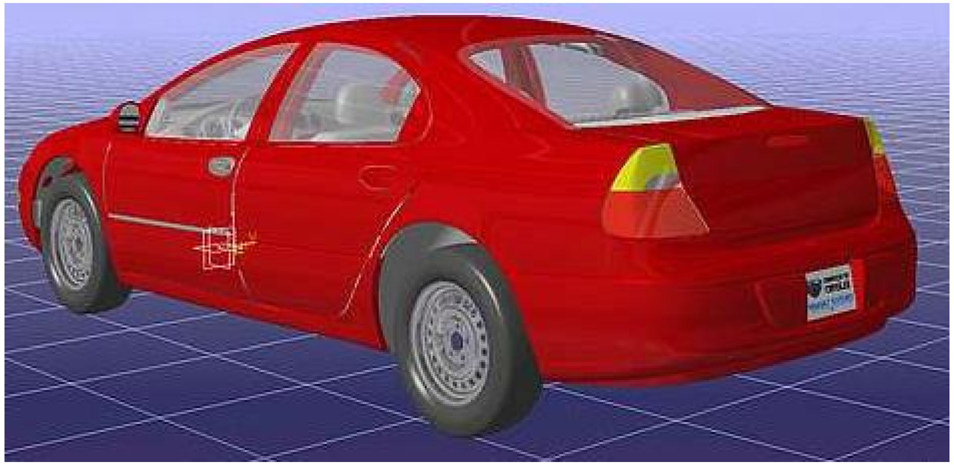
- To represent boundaries of objects exactly



Polygonal Meshes



Surface Patches

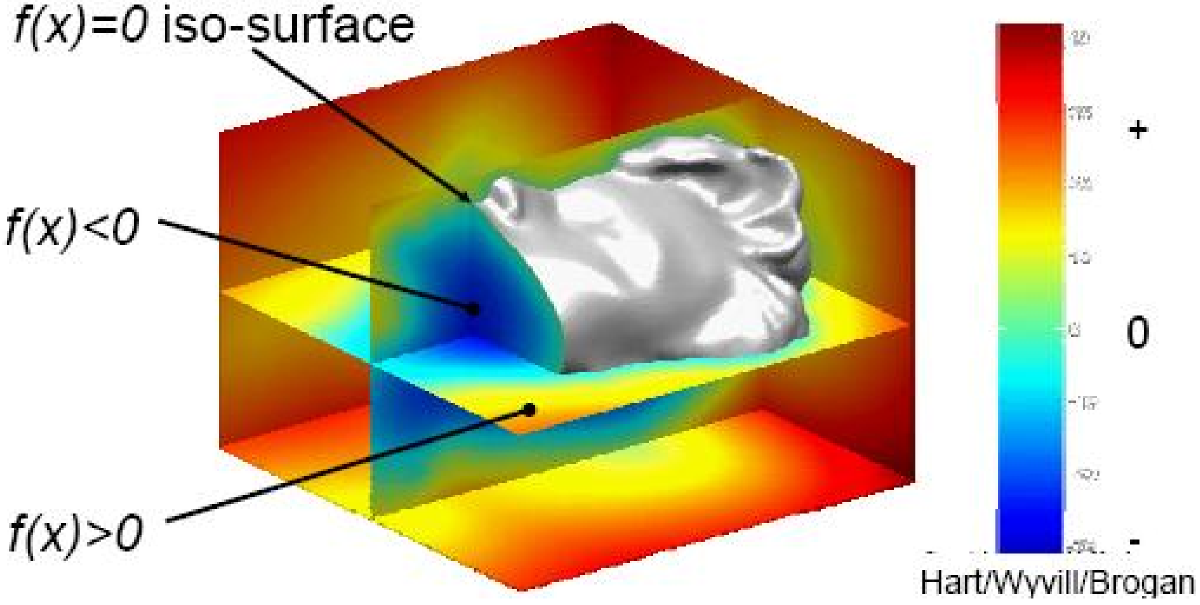


Representing Surfaces

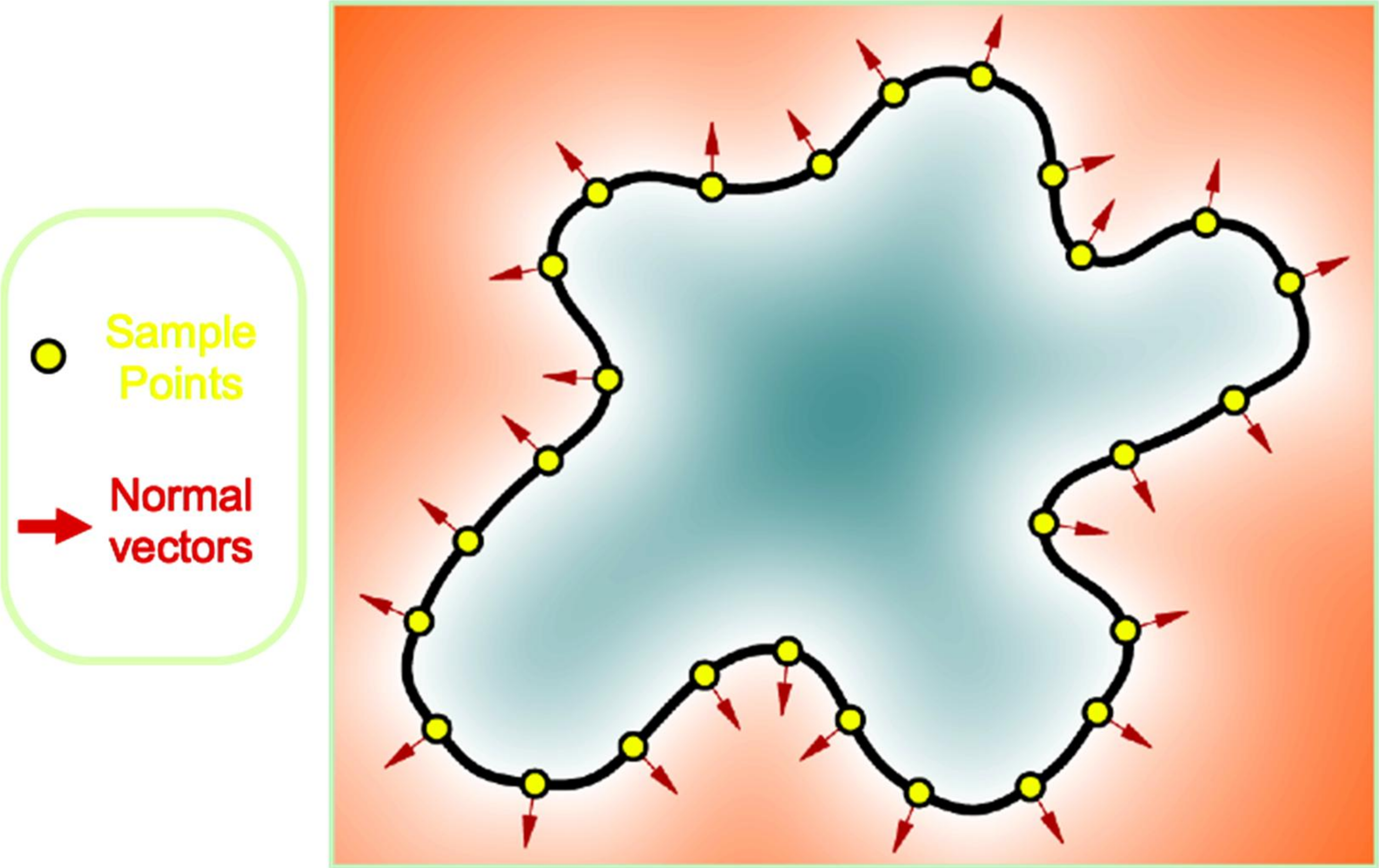
- Implicit surfaces
- Parametric surfaces
- Polygonal model

Implicit Surfaces

- Represented as: f(x, y, z) = 0
- Divide the space inside/outside of the surface based on whether f <0 or >0



Implicit Surfaces



From Shen, *et al.*, SIGGRAPH, 2004.

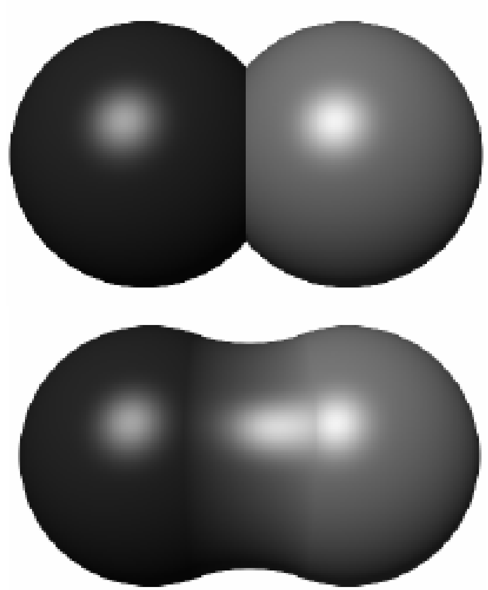
Implicit Surfaces – Blobs

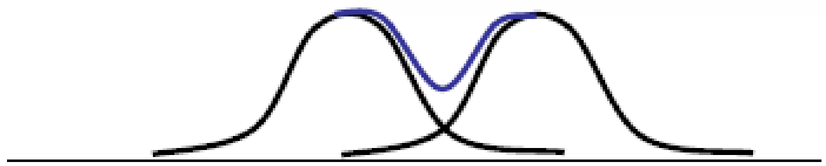
- Sum of Gaussians

$$r_i^2(x,y,z) = (x-x_i)^2 + (y-y_i)^2 + (z-z_i)^2$$

$$f(x) = -1 + \sum b_i \cdot \exp(-a_i r_i^2)$$

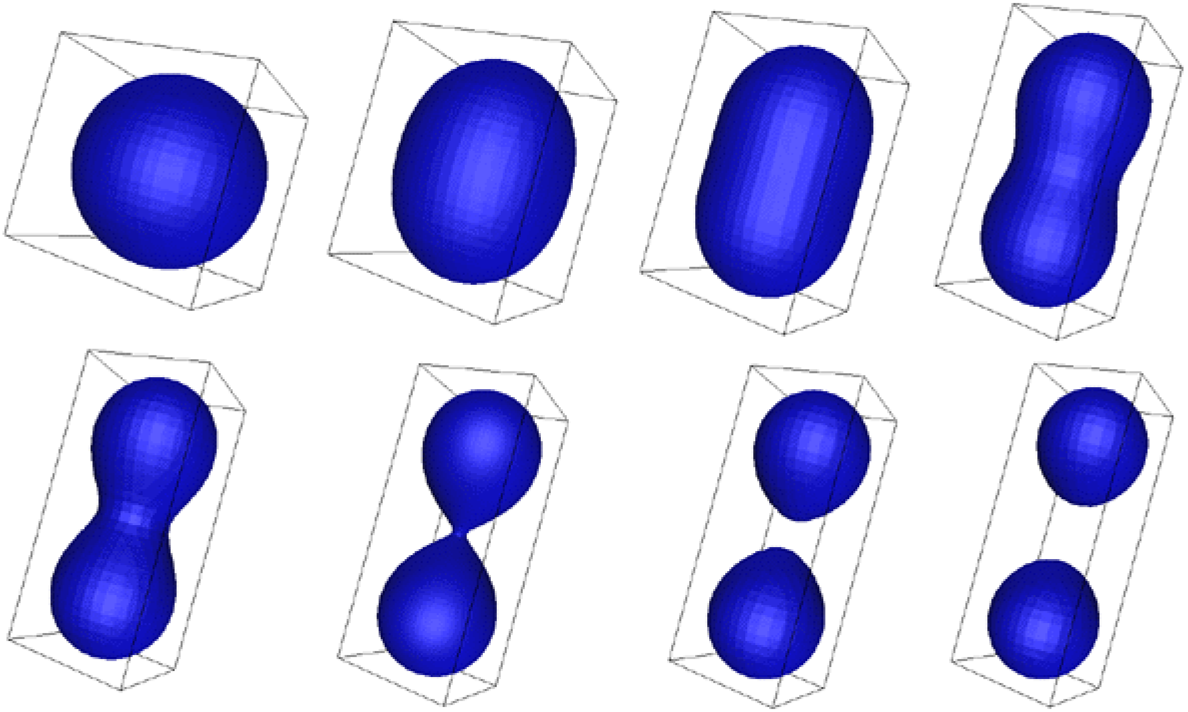
  - Points where f(x)=0 form the surface
  - a_i and b_i determine the radius and shape of each blob



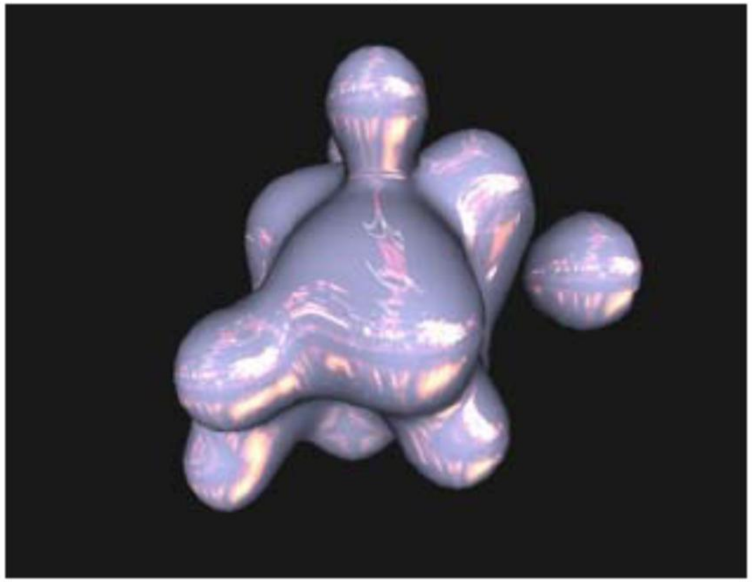


Images by A. Varshney

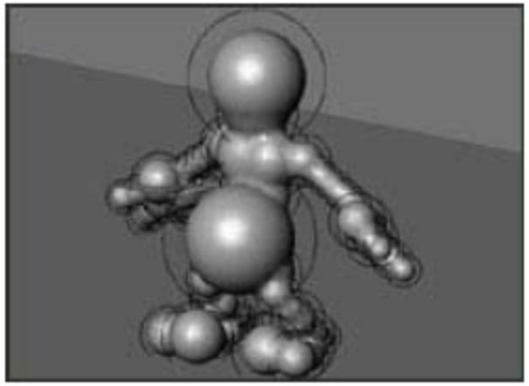
Implicit Surfaces – Blobs



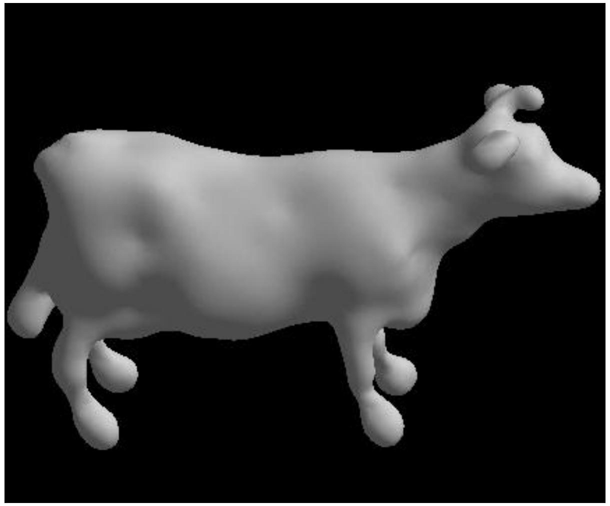
Implicit Surfaces – Examples



Yury Uralsky, NVIDIA, GDC 2006



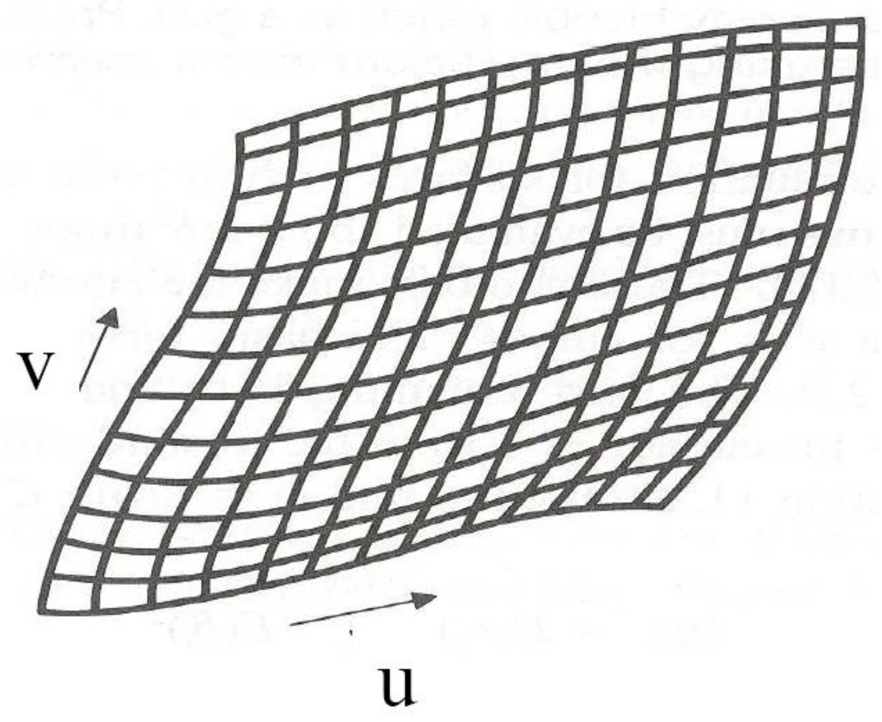
http://blender3dfr.free.fr/anglais/tut6/tut6.htm



Parametric Surfaces

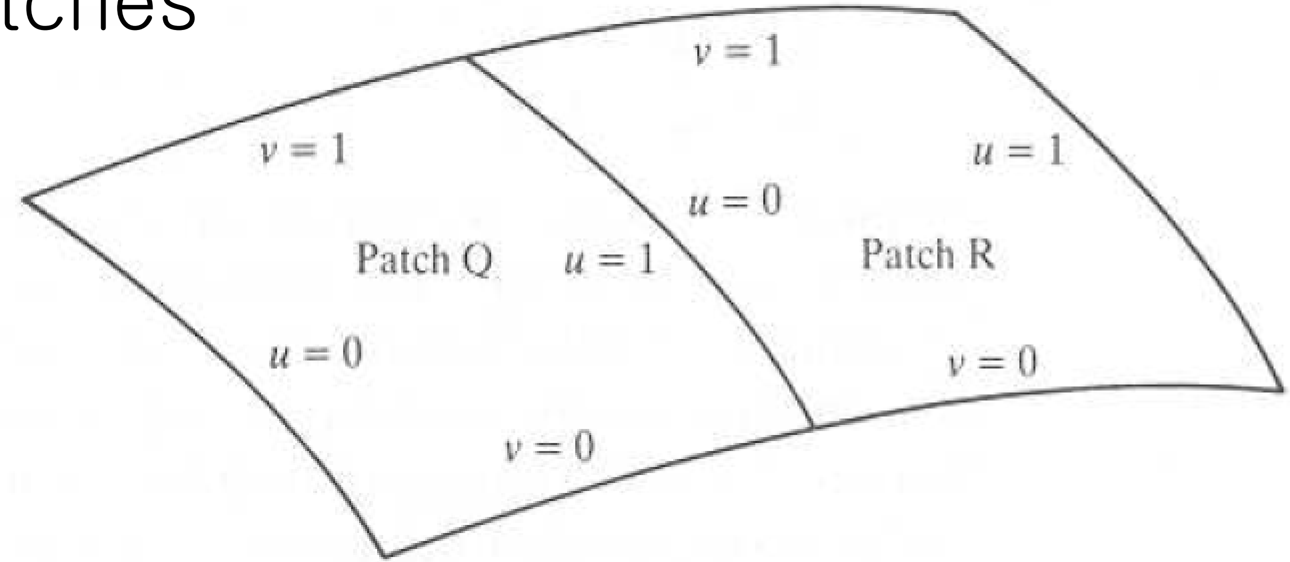
- The value of each component (x, y, z) depends on independent variables, u and v

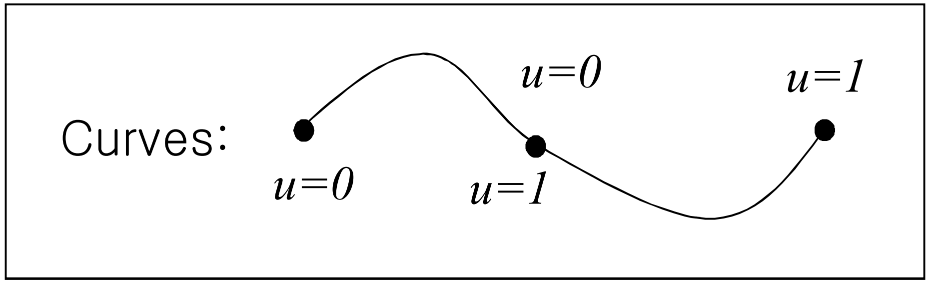
$$p(u,v)=\begin{bmatrix} x(u,v) \\ y(u,v) \\ z(u,v) \end{bmatrix}$$



Piecewise Parametric Surfaces

- Surface is partitioned into parametric patches





Parametric Surfaces

- Useful for modeling surfaces where continuity is important
- Various representations
  - Bezier
  - B-Splines
  - NURBS (Non-Uniform Rational B-Splines)

Bezier Curves

- A Bezier curve of degree n over n+1 control points $p_0, \ldots, p_n$ is defined as:

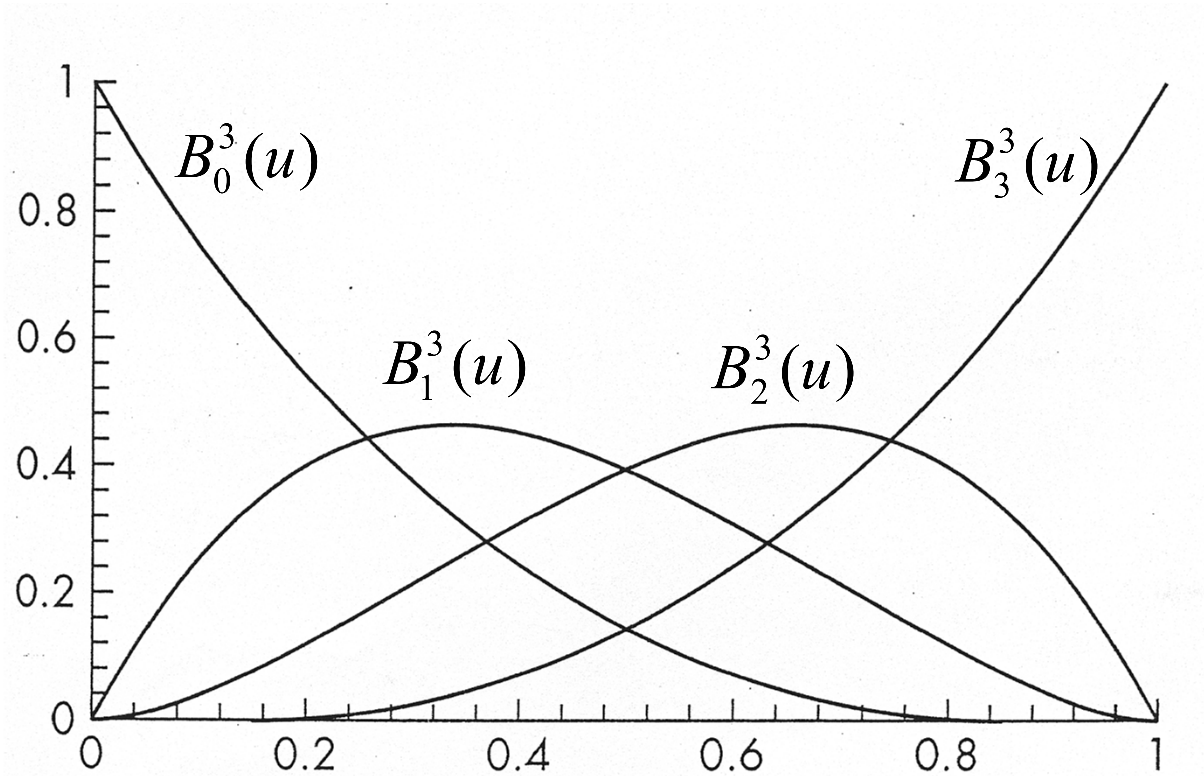
$$p(u) = \sum_{k=0}^{n} p_k B_k^n(u), \ \ 0 \le u \le 1$$

- Where, $B_k^n(u) = \binom{n}{k} u^k (1-u)^{n-k}$

$$= \frac{n!}{k!(n-k)!} u^k (1-u)^{n-k}$$

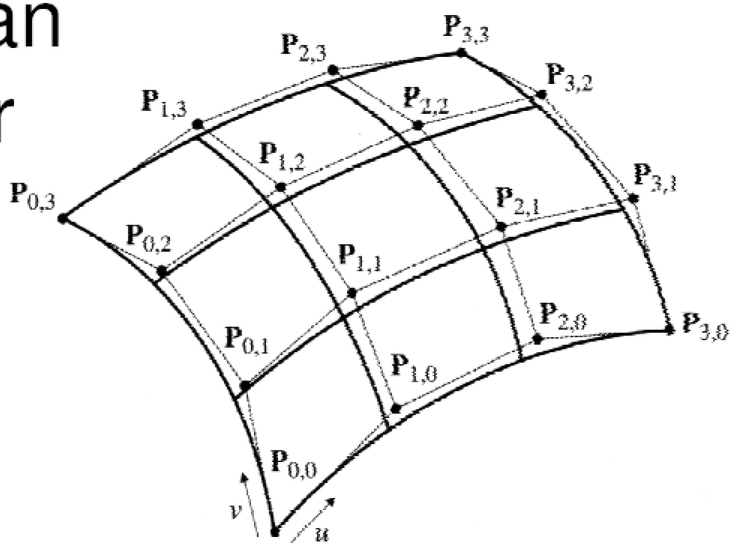
(Bernstein polynomials)

Bernstein polynomials



Bezier Surfaces

- The Bezier surface is an extension of the Bezier curve concept to one higher dimension.
- Evaluate in v to get control points in u.

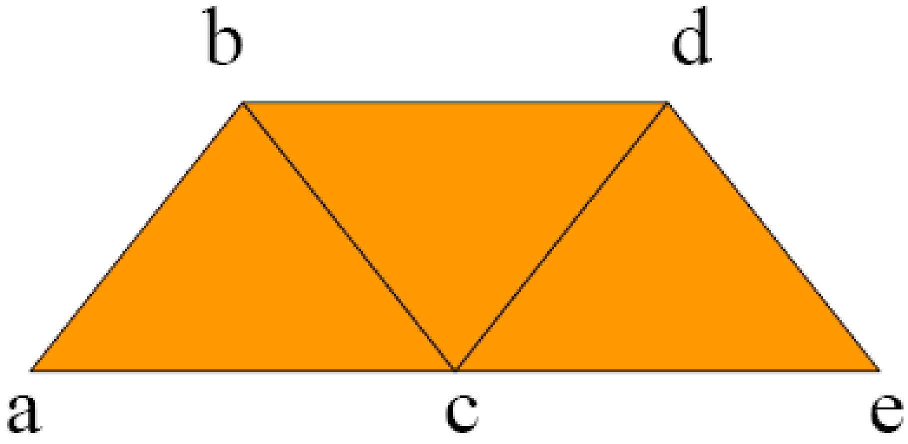


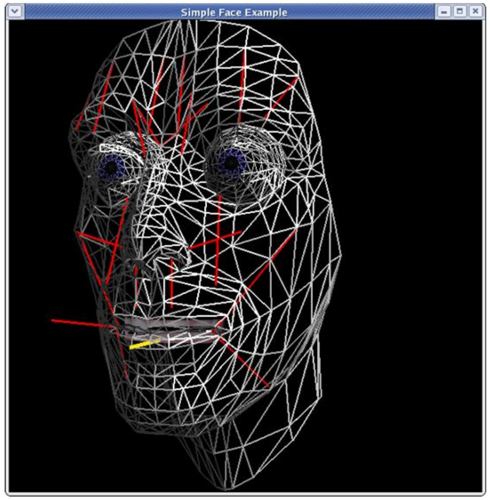
$$\mathbf{P}(u,v)=\sum_{i=0}^{n}\sum_{j=0}^{m}\mathbf{P}_{i,j}B_{i,n}(u)B_{j,m}(v) \qquad (0\le u\le 1, 0\le v\le 1)$$

$$\mathbf{P}(u,v)=\sum_{i=0}^{n}[\mathbf{P}_{i,0}B_{0,m}(v)+\mathbf{P}_{i,1}B_{1,m}(v)+\cdots+\mathbf{P}_{i,m}B_{m,m}(v)]B_{i,n}(u)$$

Polygonal Model

- Triangle mesh representation

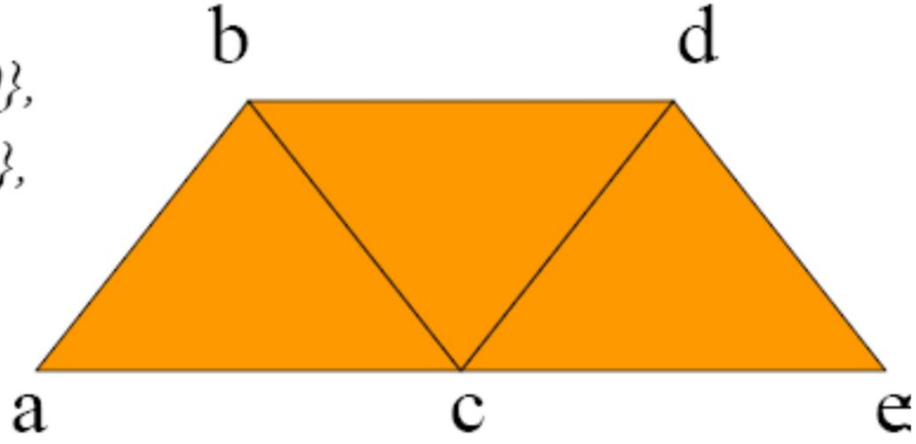




- **Geometry:** *{x, y, z}* coordinates of vertices *a* .. *e*
- **Connectivity:** Triangles *abc, bcd, cde*

Direct Mesh Representation

Let $a = \{0, 0, 0\}$, $b = \{1, 1, 0\}$,
$c = \{2, 0, 0\}$, $d = \{3, 1, 0\}$,
$e = \{4, 0, 0\}$



Simplest File Format

t 3 // # triangles = 3

0, 0, 0, 1, 1, 0, 2, 0, 0 // abc

1, 1, 0, 2, 0, 0, 3, 1, 0 // bcd

2, 0, 0, 3, 1, 0, 4, 0, 0 // cde

Indexed Mesh Representation

v 5 // #vertices = 5

0 0 0

1 1 0

2 0 0

3 1 0

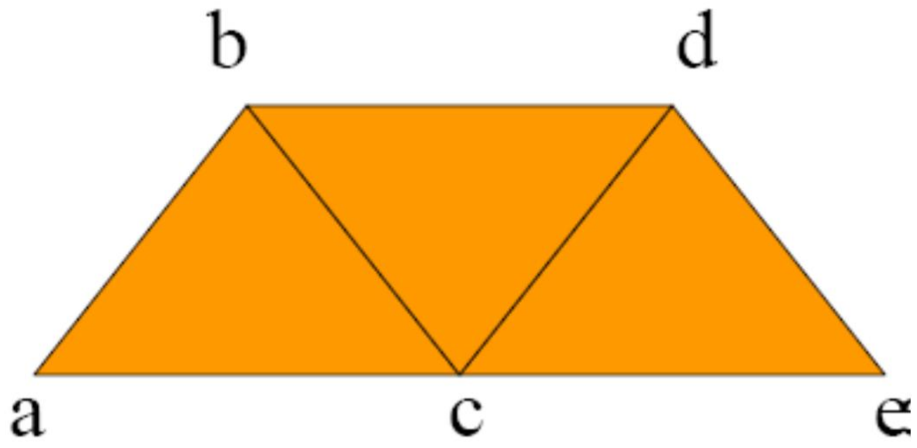
4 0 0

t 3 // # triangles = 3

0 1 2 // abc

1 2 3 // bcd

2 3 4 // cde



Indexed Mesh Representation

v 5 // #vertices = 5

0 0 0

1 1 0

2 0 0

3 1 0

4 0 0

c 2 // # colors = 2

255 0 0 // red in byte rep

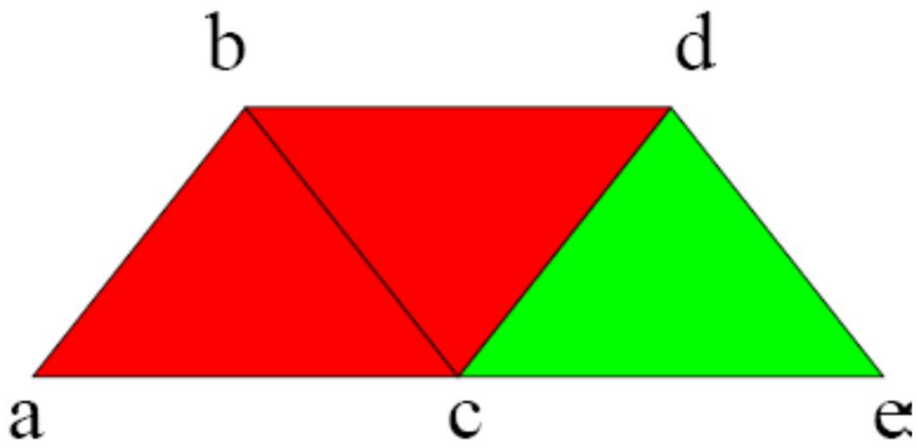
0 255 0 // green in byte rep

t 3 // # triangles = 3

0 0 1 0 2 0 // abc

1 0 2 0 3 0// bcd

2 1 3 1 4 1// cde

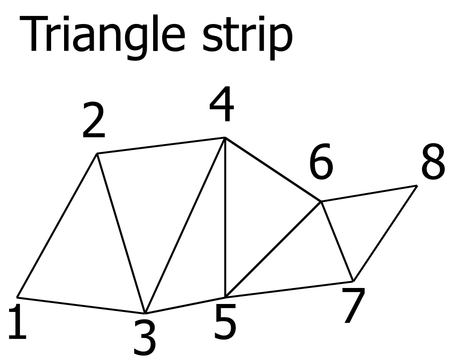


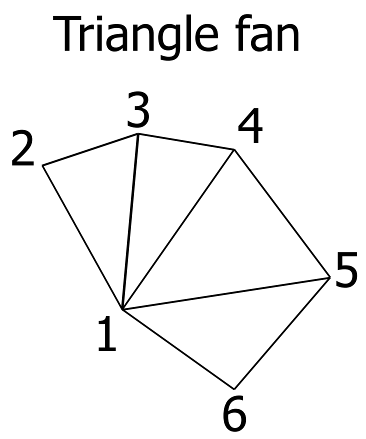
Model Transmission Acceleration

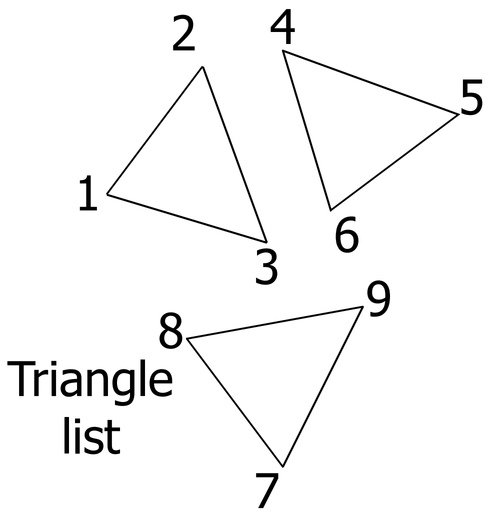
- Typically, triangle list is transmitted to GPU memory from system memory, in each frame
- Acceleration by reducing the amount of transmission
  - Triangle Strips
  - Display List
  - Vertex Array or Vertex Buffer

Triangle Strips/Fan

- Triangle strips/fan can reduce the number of vertices transmitted to GPU







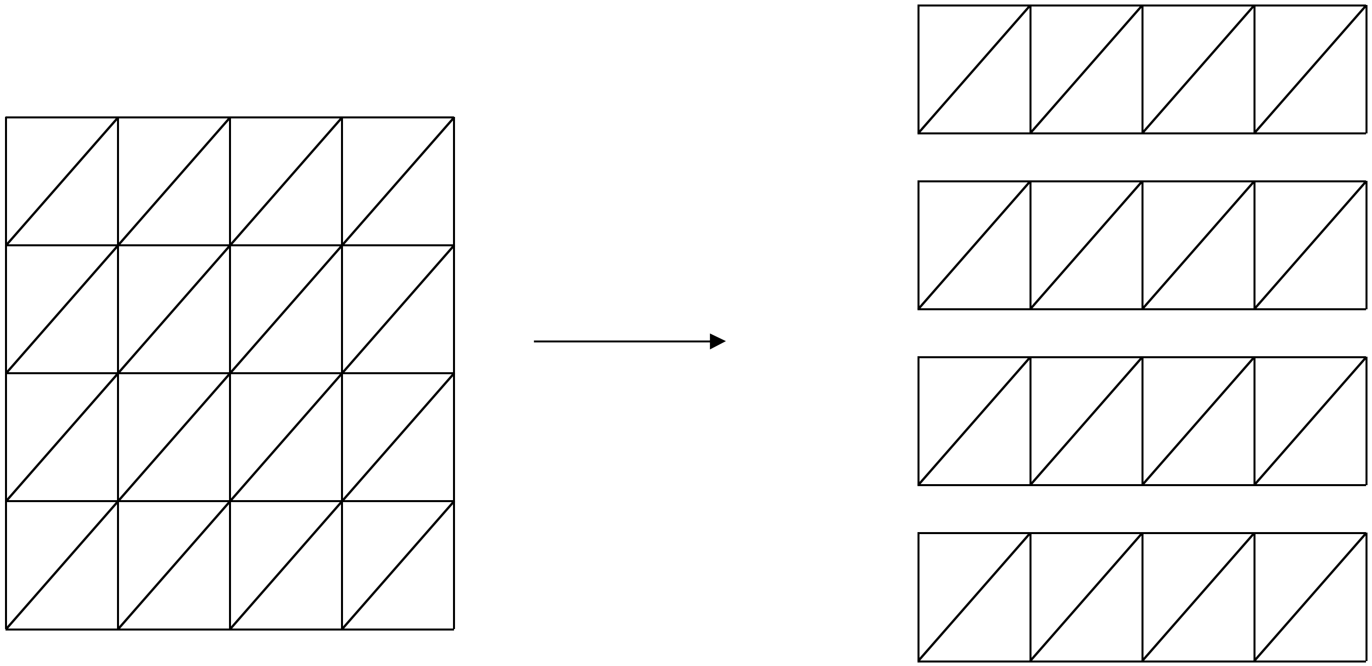
Triangle Strips/Fan

- List has no sharing
  - Vertex count = triangle count * 3
- Strips and fans share adjacent vertices
  - Vertex count = triangle count + 2
  - Lower memory
  - Topology restrictions
  - Have to break into multiple rendering calls

Triangle Strips/Fan

- Most meshes: tri count = 2x vert count
- Using lists duplicates vertices a lot!
  - Total of 6x number of rendering verts
- Strips or fans still duplicate vertices
  - Each strip/fan needs its own set of vertices
  - More than doubles vertex count
    - Typically 2.5x with good strips
  - Hard to find optimal strips and fans
  - Have to submit each as separate rendering call

Triangle Strips/Fan



32 triangles, 25 vertices

4 strips, 40 vertices

25 to 40 vertices is 60% extra data!

Display List

- Define a series of OpenGL commands as a display list
- Just call the display list to render it
- Display list is stored in GPU memory (fast!)

- Define:
  - glNewList (...) ... glEndList()
- Call: glCallList (*display_list_id*)

Indexed Primitives

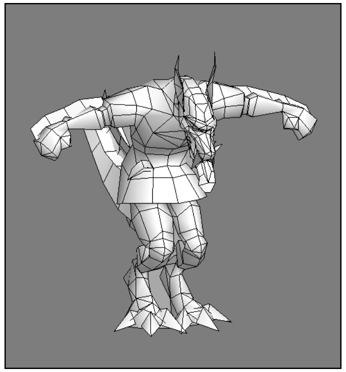
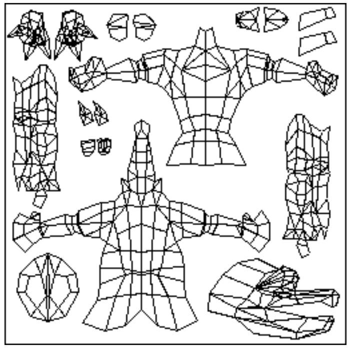
- Vertices stored in separate array
  - No duplication of vertices
  - Called a “vertex buffer” or “vertex array”
- Triangles hold indices, not vertices
- Index is just an integer
  - Typically 16 bits
  - Duplicating indices is cheap
  - Indexes into vertex array

Modeling for Games

- The most popular: Triangle mesh
  - Very straightforward
  - Easy to troubleshoot
  - Easy to modify
  - Supported by all 3D game engines and H/W
- Characteristics of models for games
  - Low polygon counts
  - Extensive use of textures for detail
  - Visibility Culling

Low Polygon Count Modeling

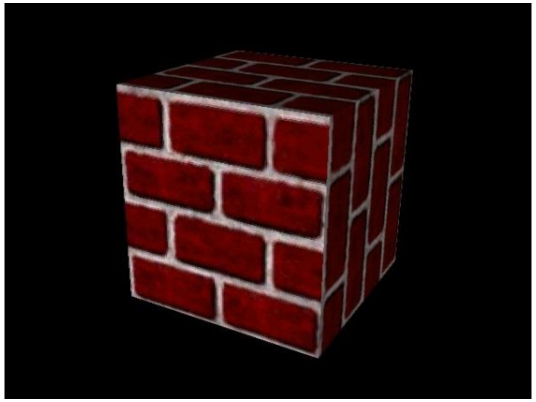
- With low polygon modeling, much of the detail is painted into the texture
- Faceting
  - Rough around the edges

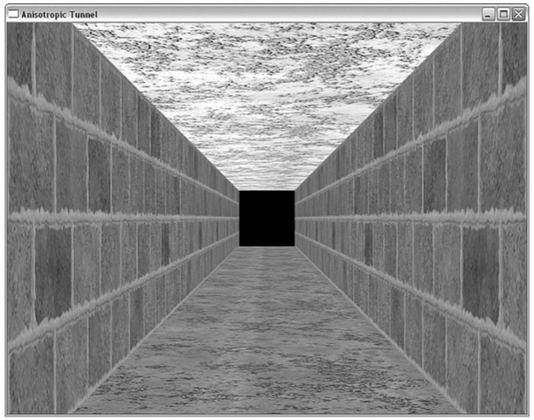
   

Images courtesy of WildTangent, model and texture by David Johnson.

Normal Mapping

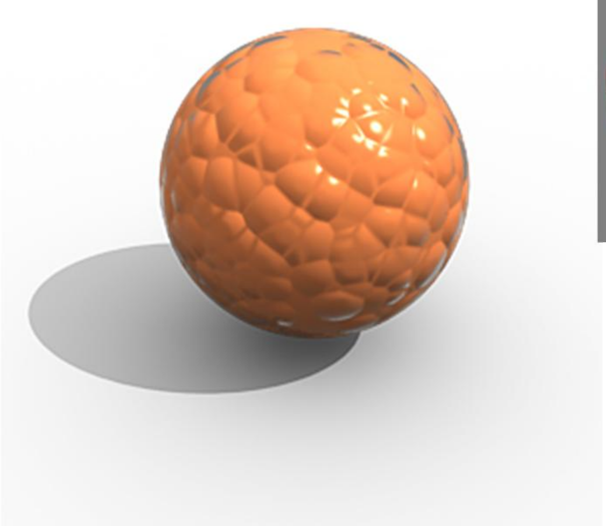
- Textured surfaces look flat

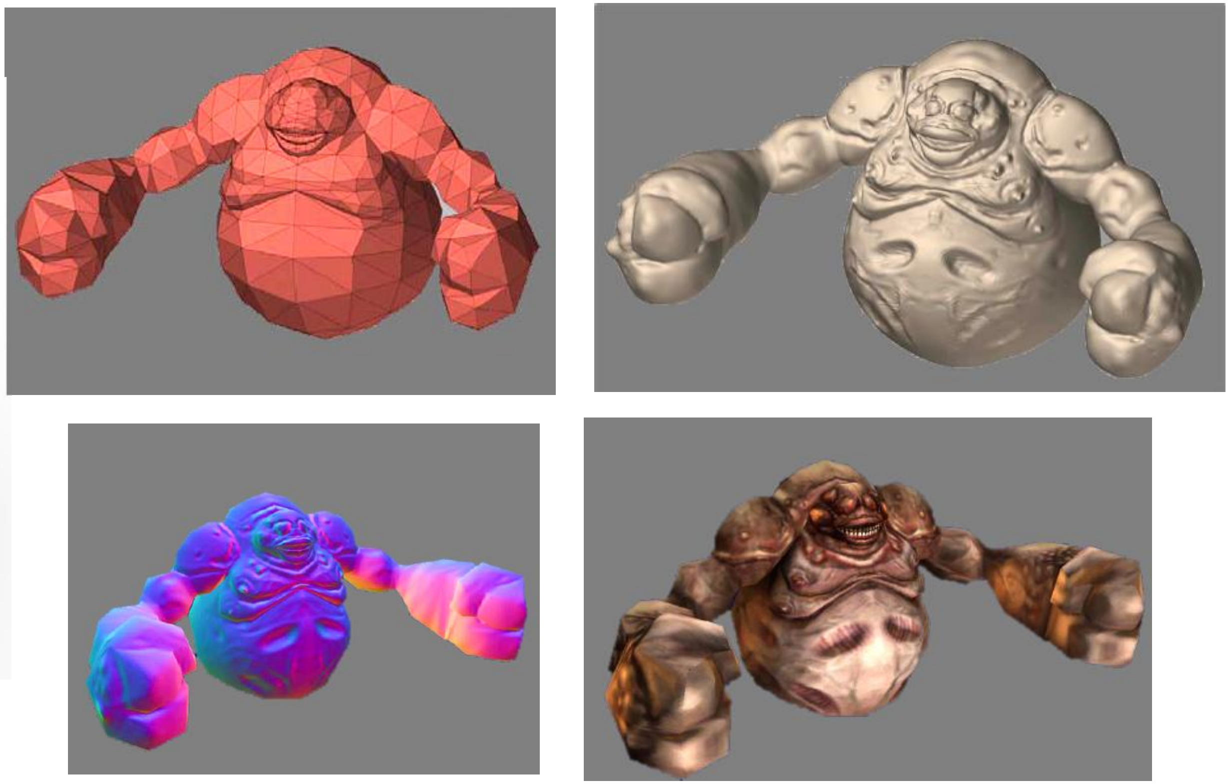




Normal Mapping

- Normal vectors of detailed texture are stored in normal maps

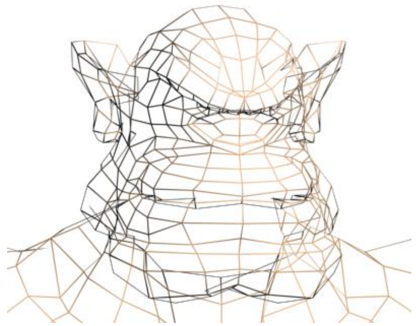




Images courtesy of Pixolgic.

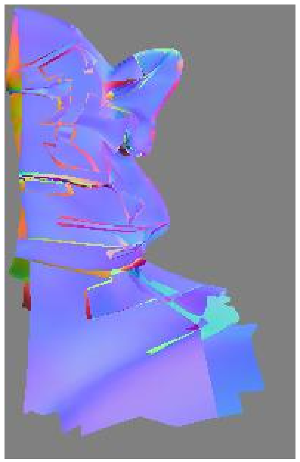
Normal Mapping

- Low-polygon model

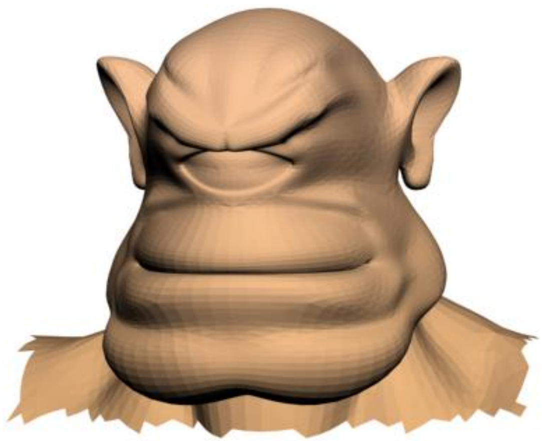


- Normal Map

Result

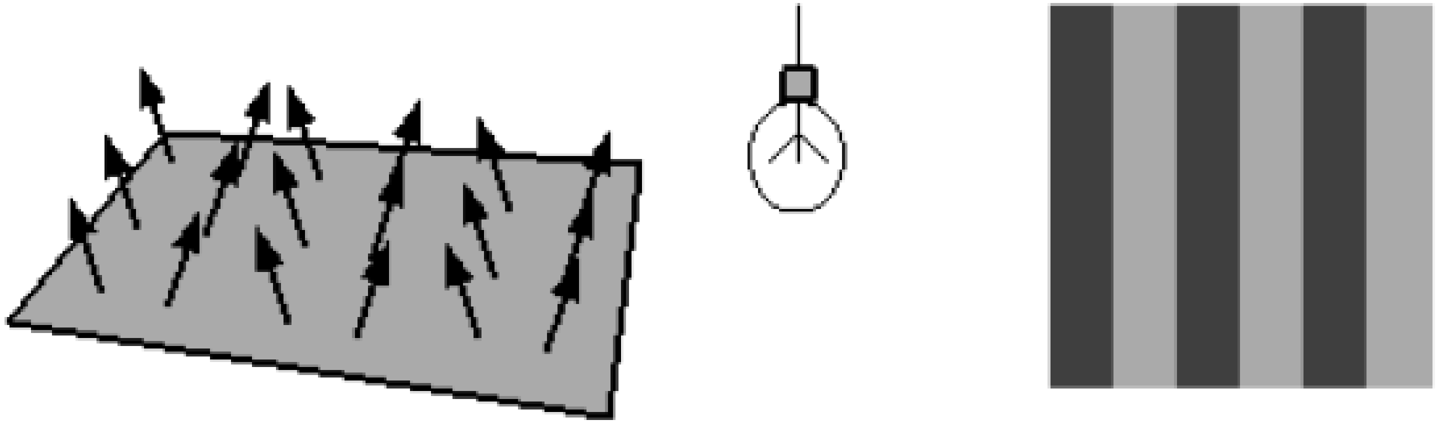


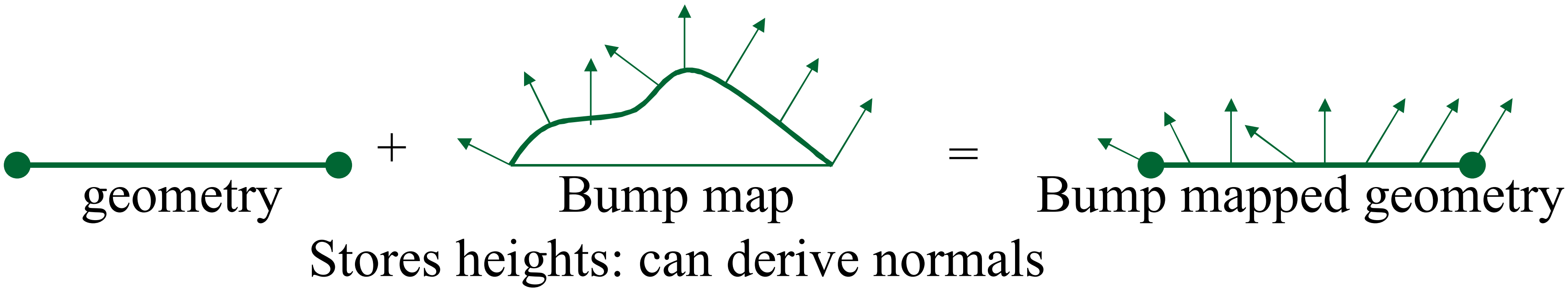
Normal map is an RGB image where the color value (*R*, *G*, *B*) of each pixel stores the (*X*, *Y*, *Z*) coordinates of a surface normal for faking the lighting of bumps and dents



Bump Mapping

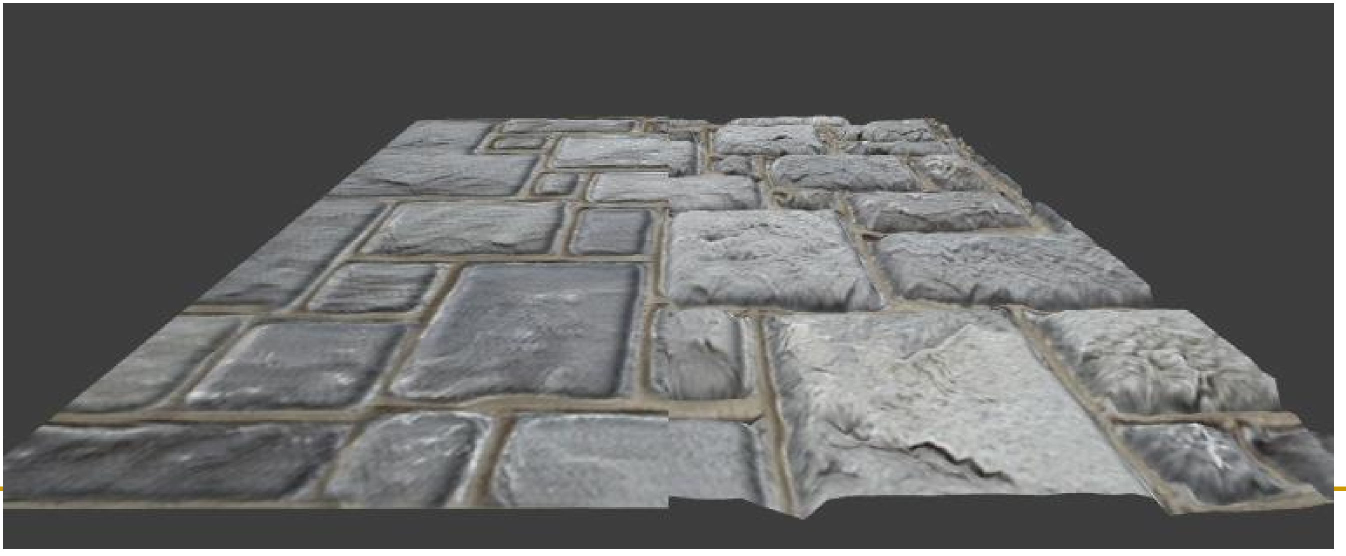
- Texture values perturb surface normals





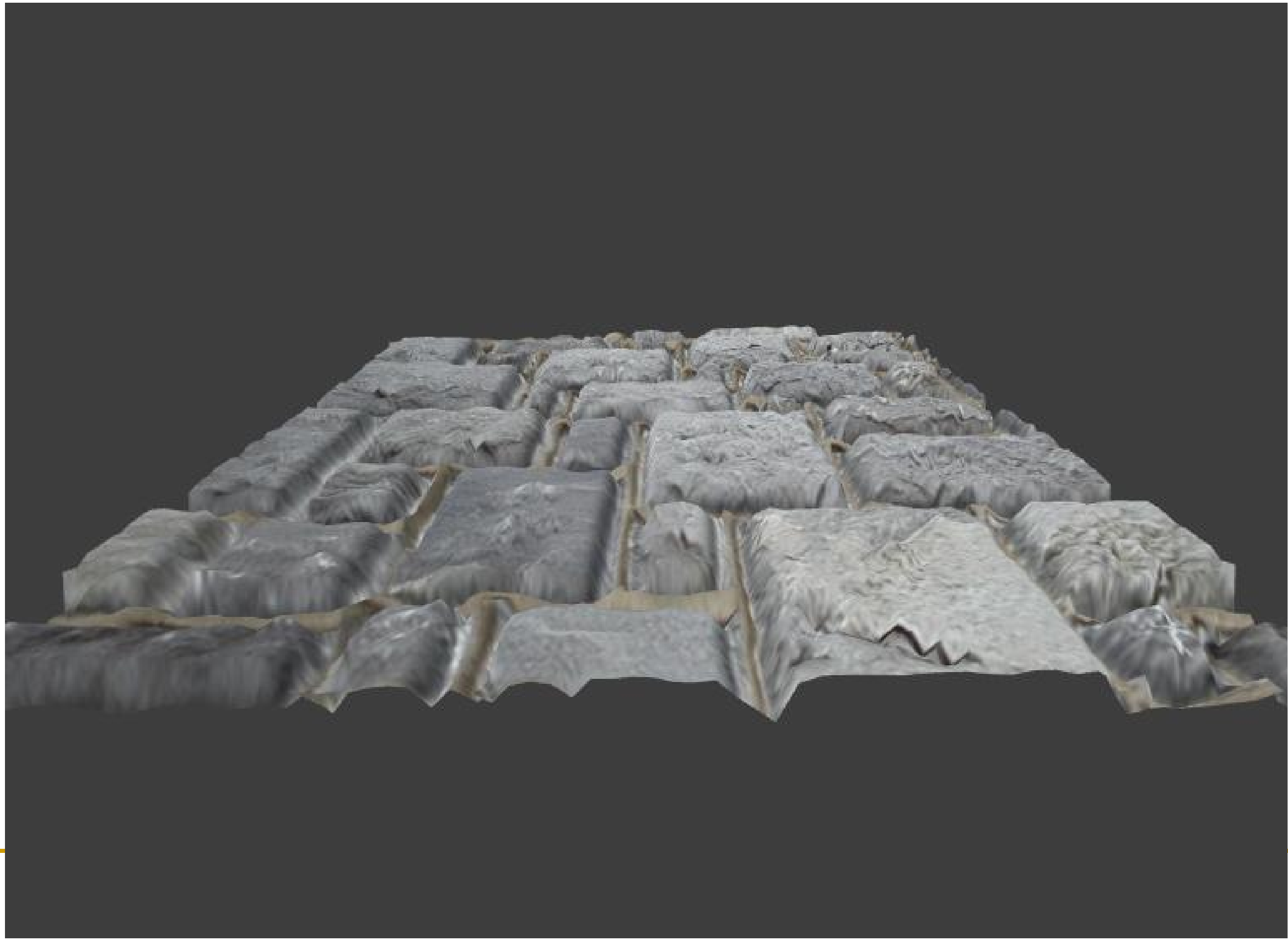
Displacement Mapping

- Normal Mapping Problem
  - Doesn't take into account geometric surface depth
    - Does not exhibit **parallax**
    - No self-shadowing of the surface
    - Coarse silhouettes expose the actual geometry being drawn
- Displacement Mapping
  - Displace actual positions from Heightfield Map

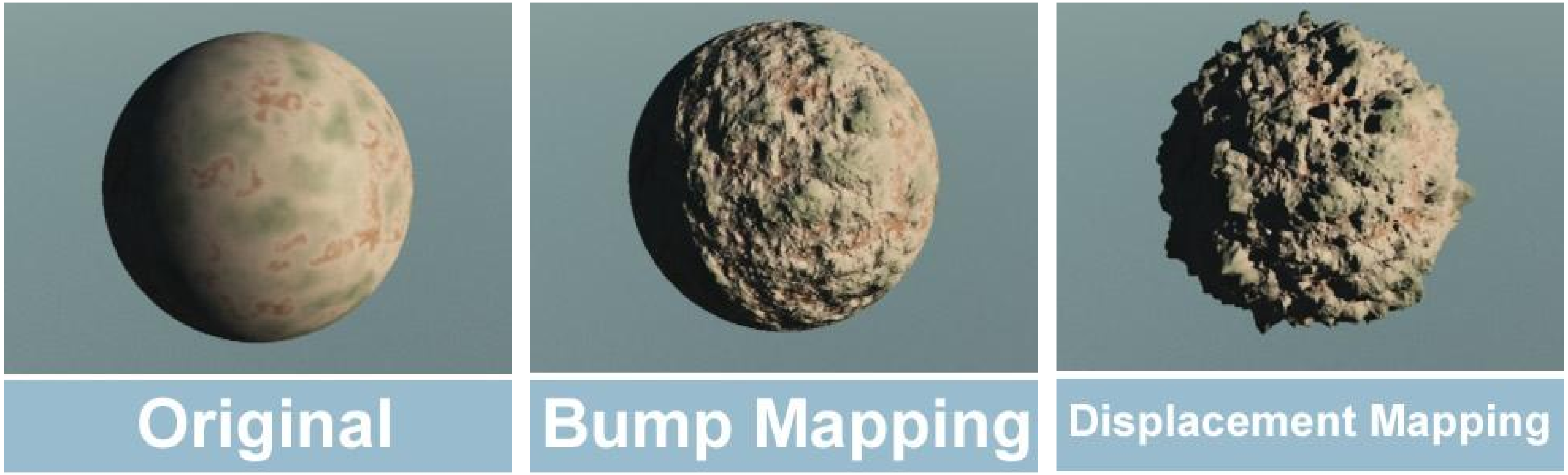


Displacement Mapping (Result)

- Displacement Offset

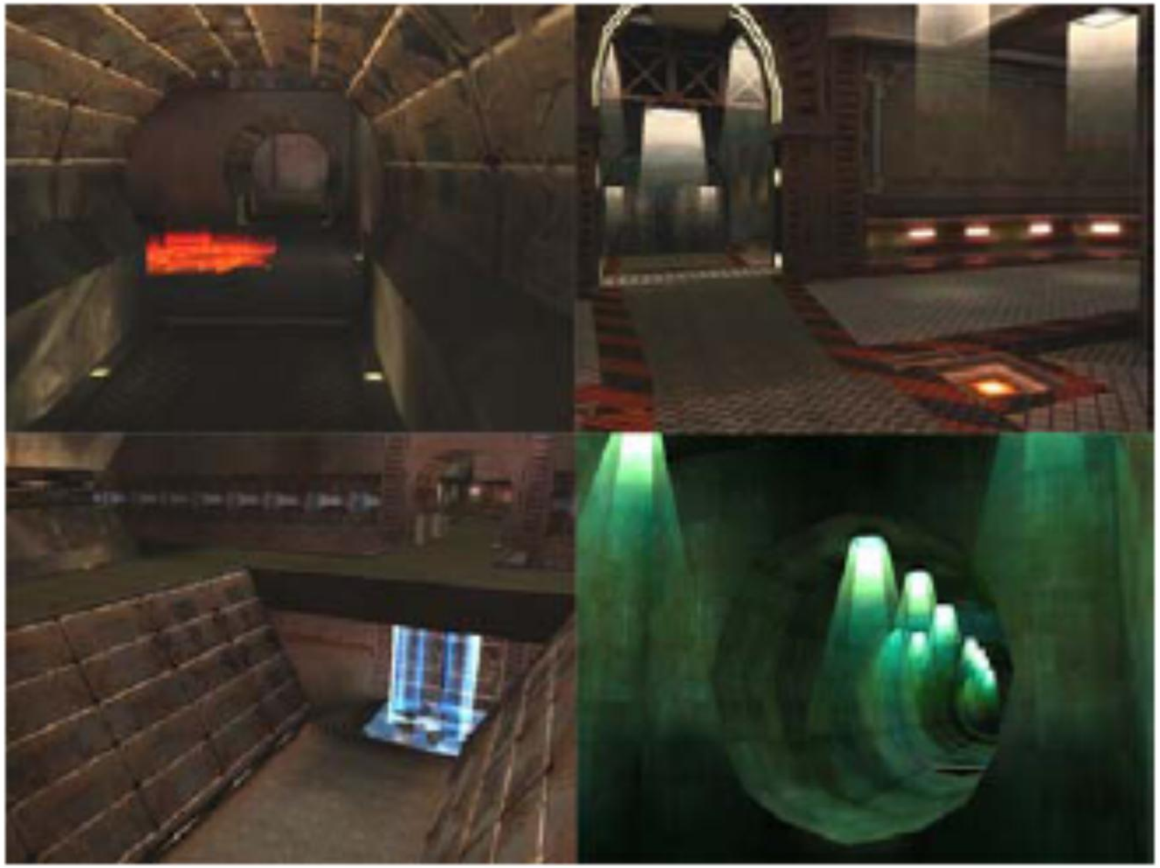


Displacement Mapping



Visibility Culling

- Architectural models or dungeons



Visibility Culling

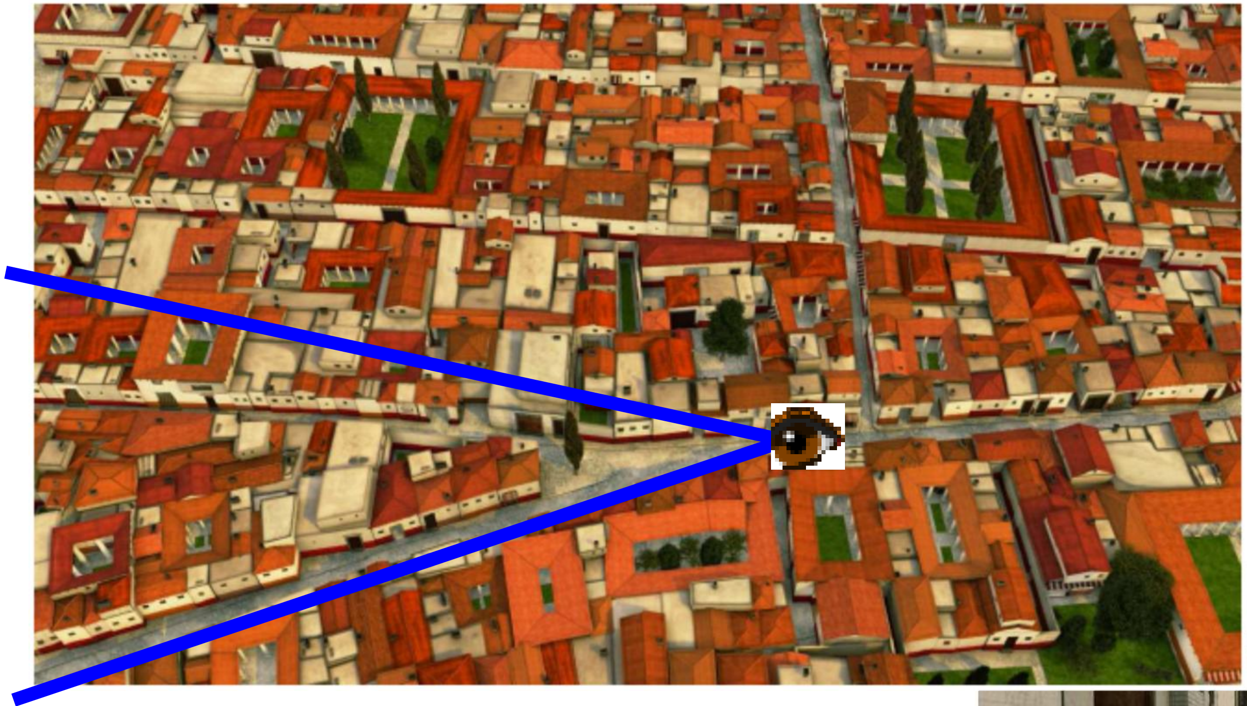


Image courtesy of Muller *et al*. SIGGRAPH 06

Visibility Culling

- Cannot draw entire world every frame
  - Lots of objects – far too slow
- Need to decide quickly what is visible
- Partition world into areas
- Decide which areas are visible
- Draw things in each visible area
- Many ways of partitioning the world

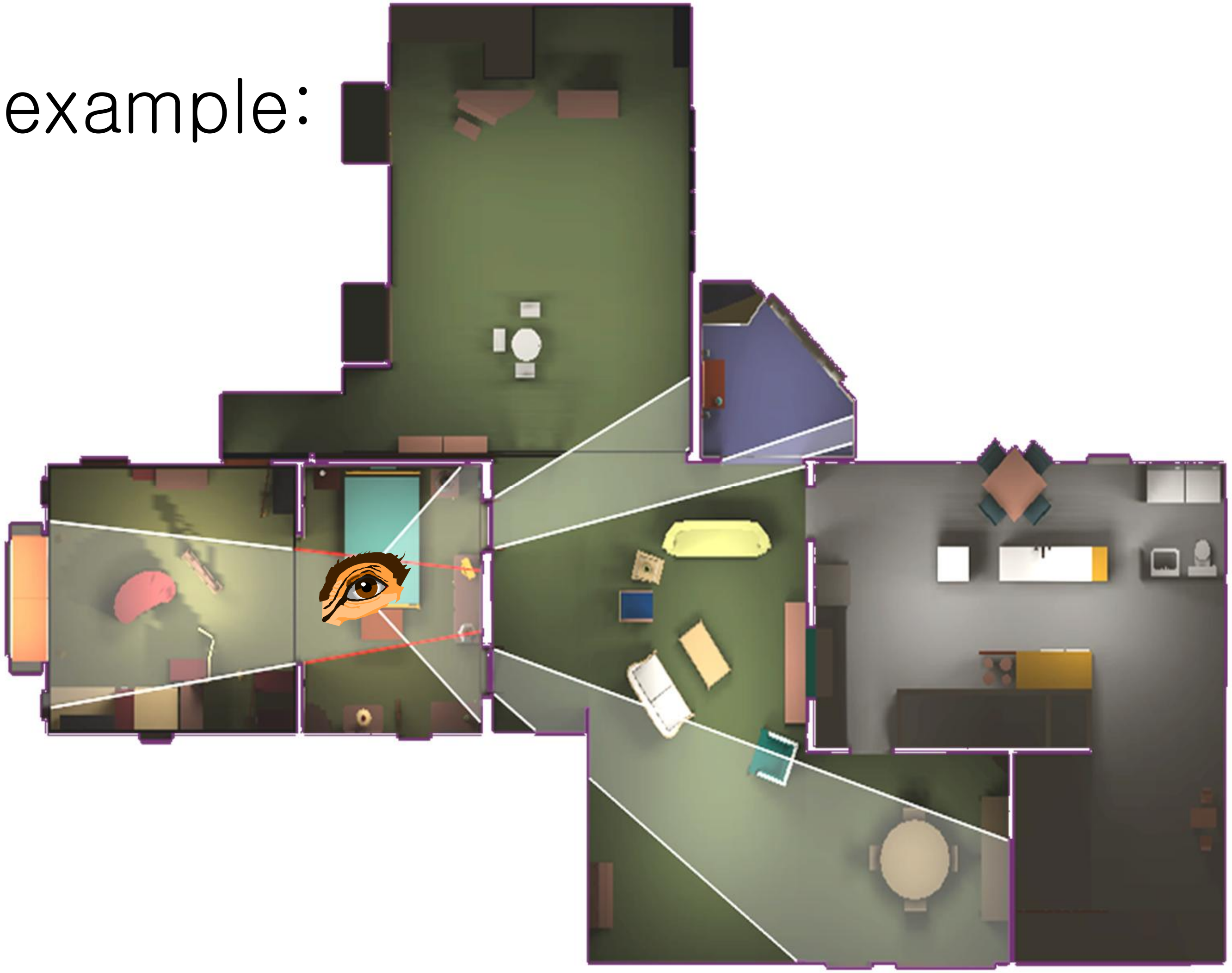
Cells & Portals

- Goal: walk through architectural models (buildings, cities, catacombs)
- These divide naturally into *cells*
  - Rooms, alcoves, corridors...
- Transparent *portals* connect cells
  - Doorways, entrances, windows...

- Notice: cells only see other cells through portals

Cells & Portals

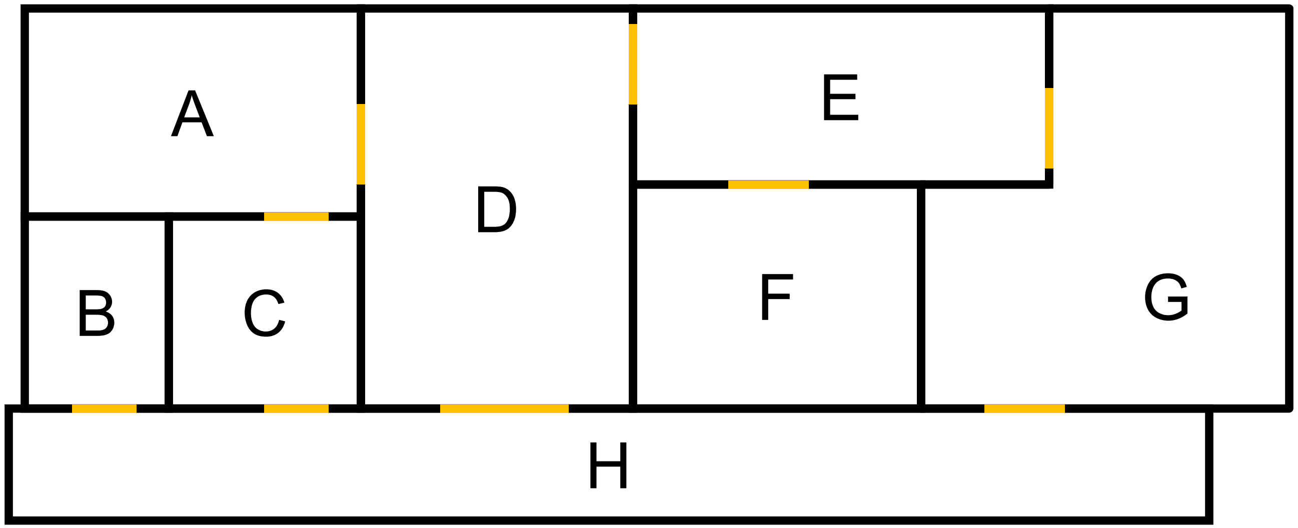
- An example:

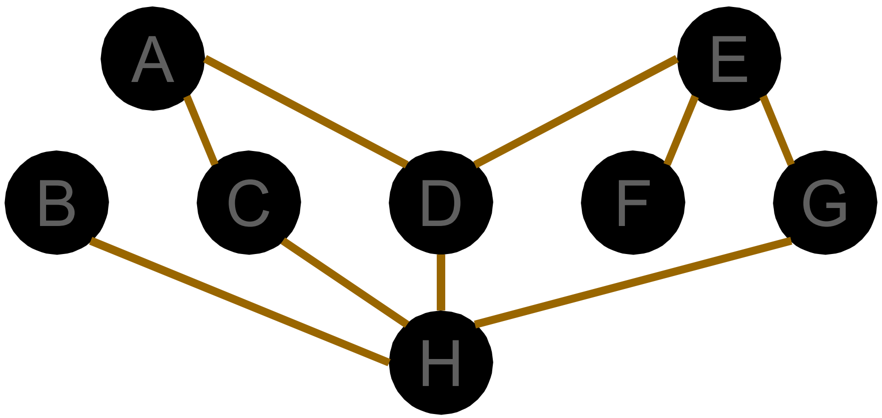


Cells & Portals

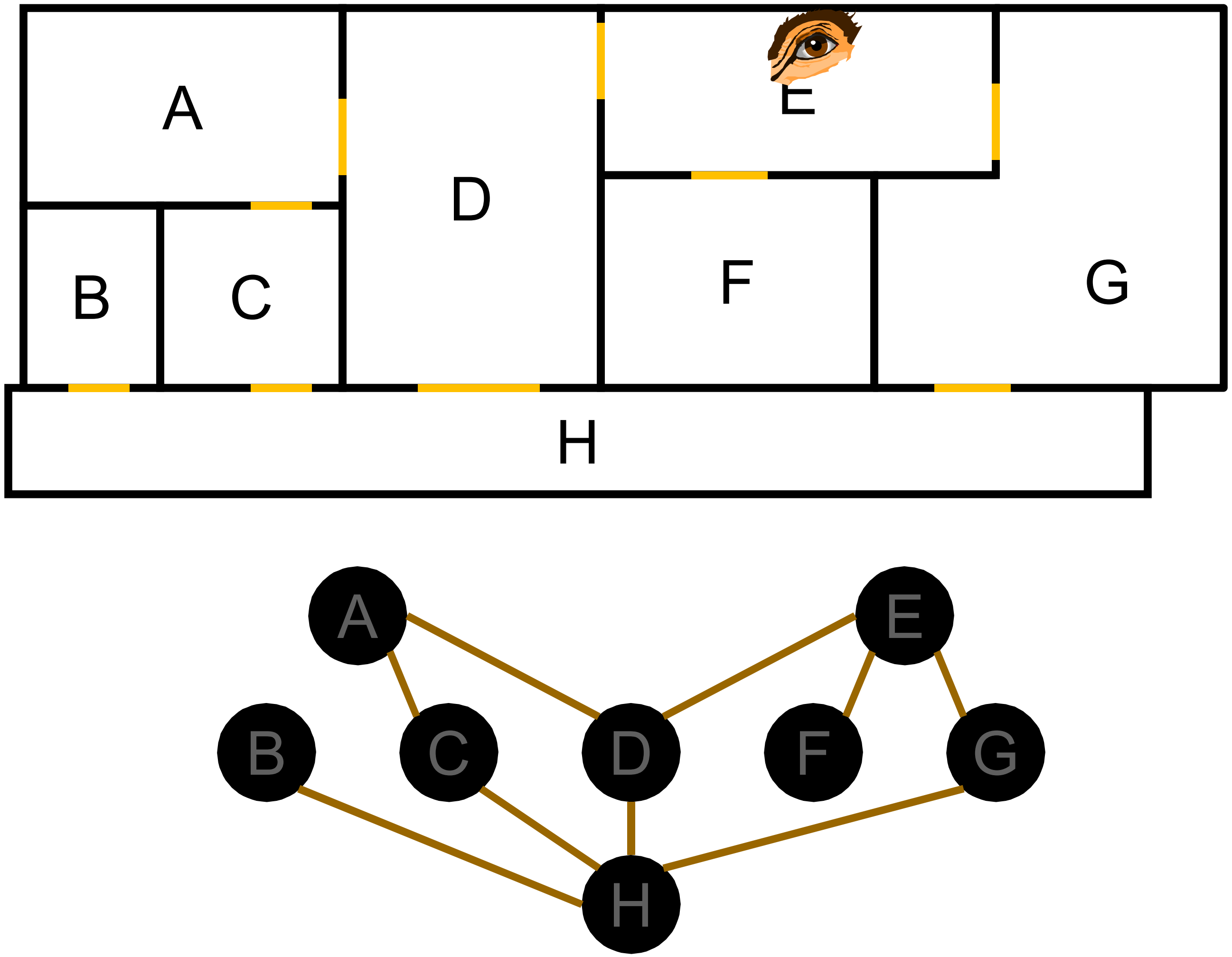
- Idea:
  - Cells form the basic unit of PVS (Potentially Visible Set)
  - Create an *adjacency graph* of cells
  - Starting with cell containing eyepoint, traverse graph, rendering visible cells
  - A cell is only visible if it can be seen through a sequence of portals
    - So cell visibility reduces to testing portal sequences for a *line of sight…*

Cells & Portals

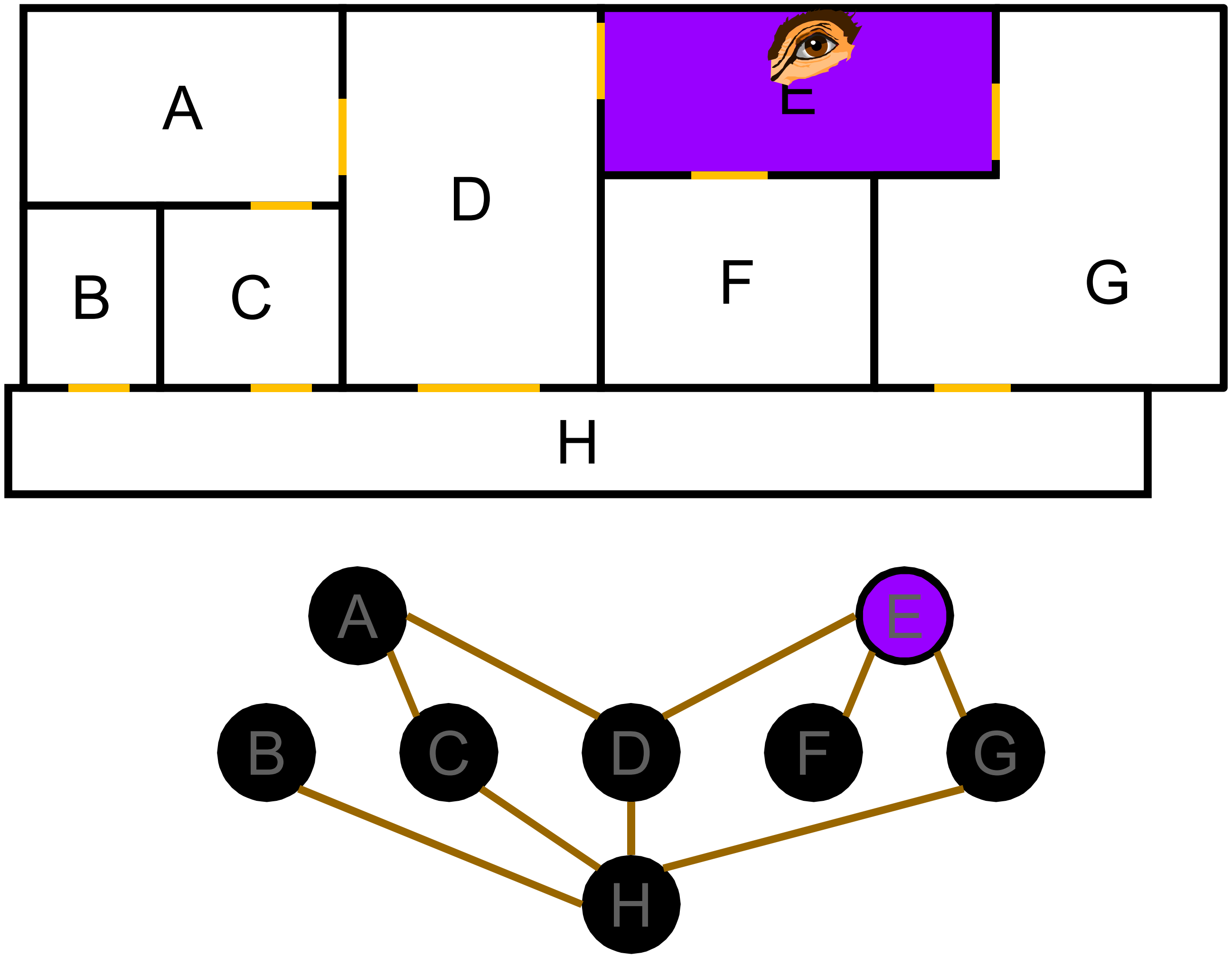




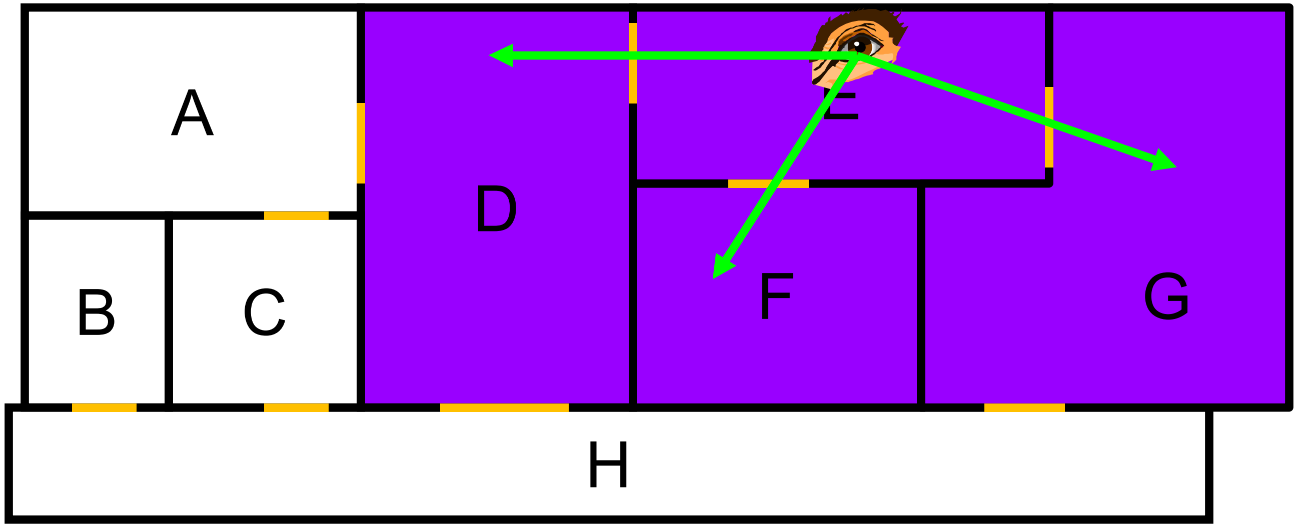
Cells & Portals

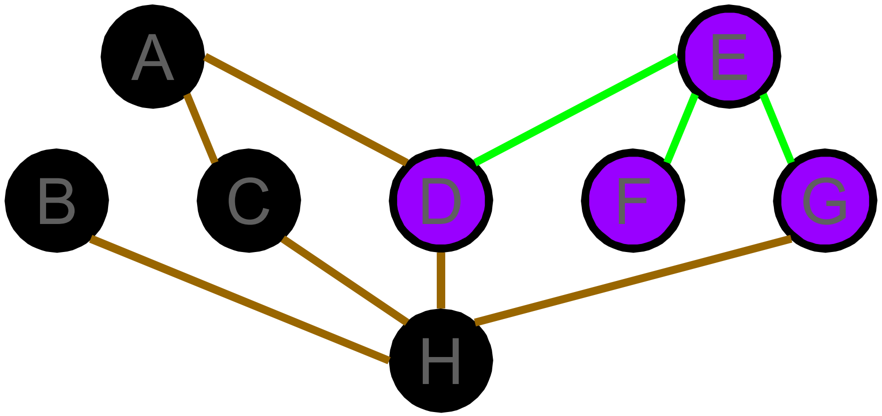


Cells & Portals

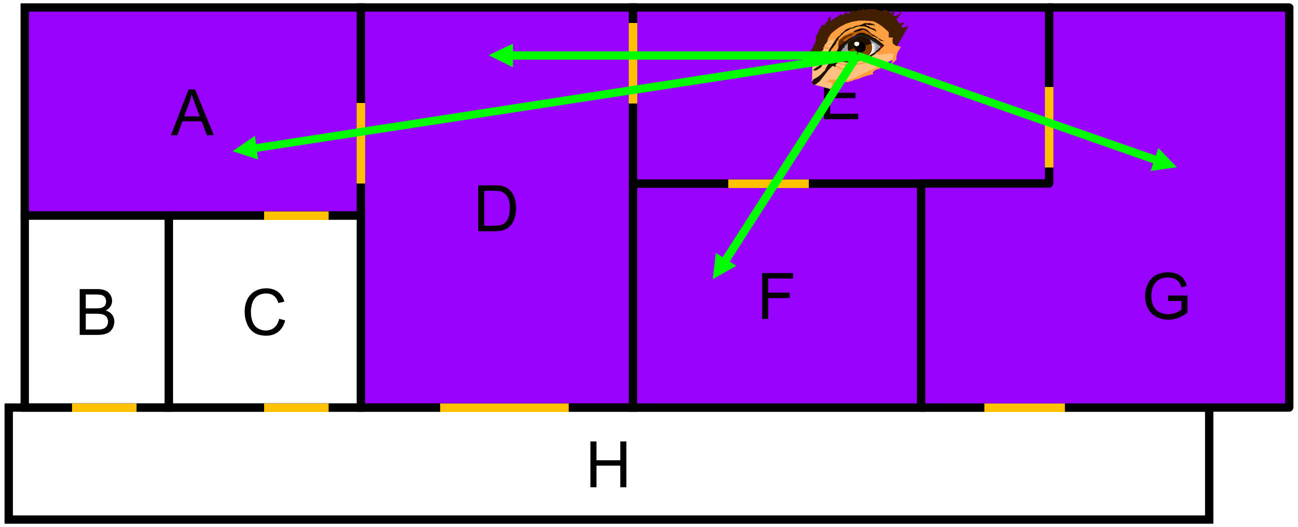


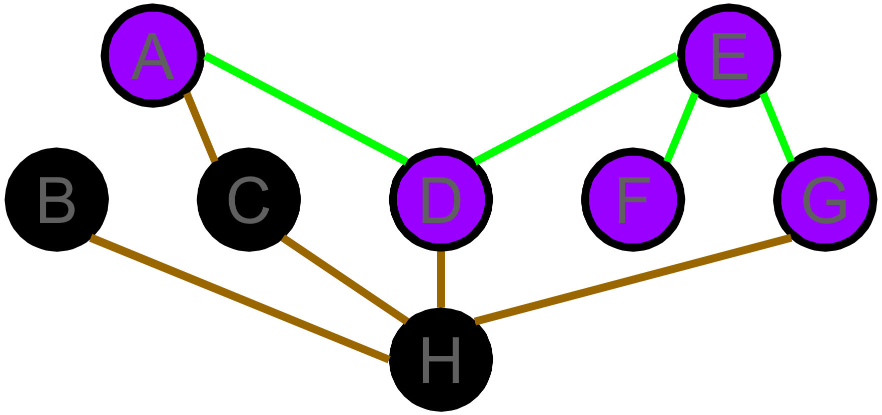
Cells & Portals



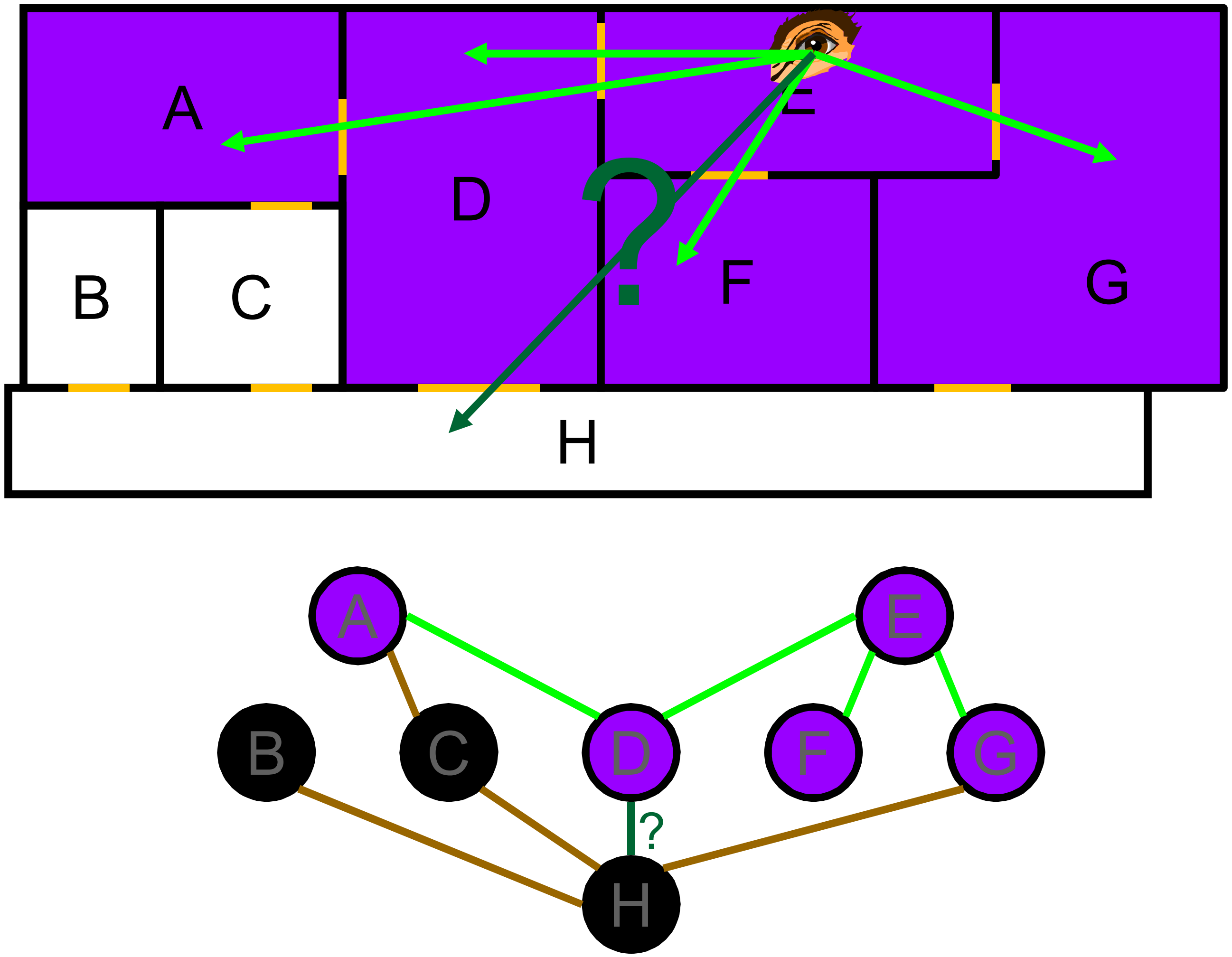


Cells & Portals

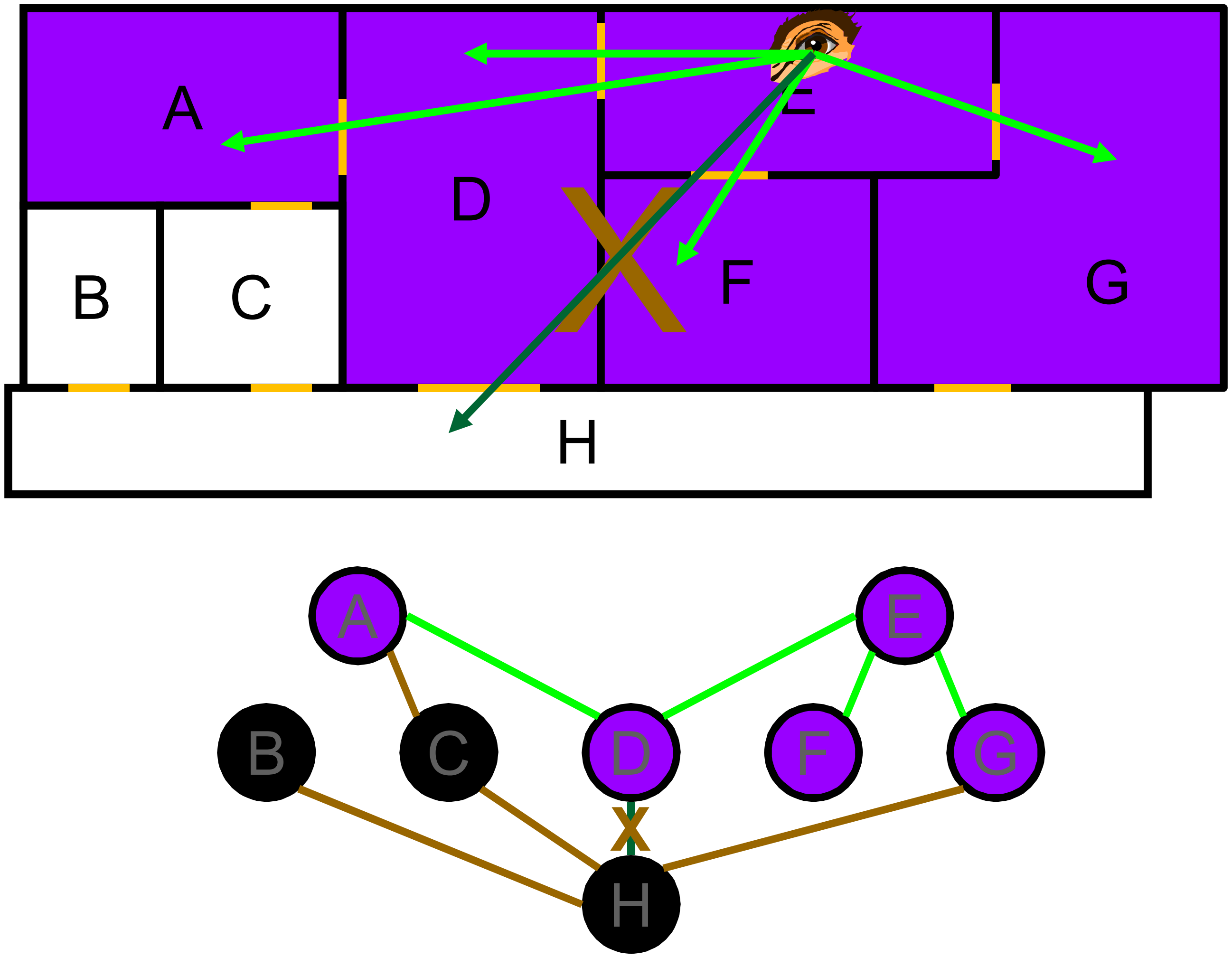




Cells & Portals

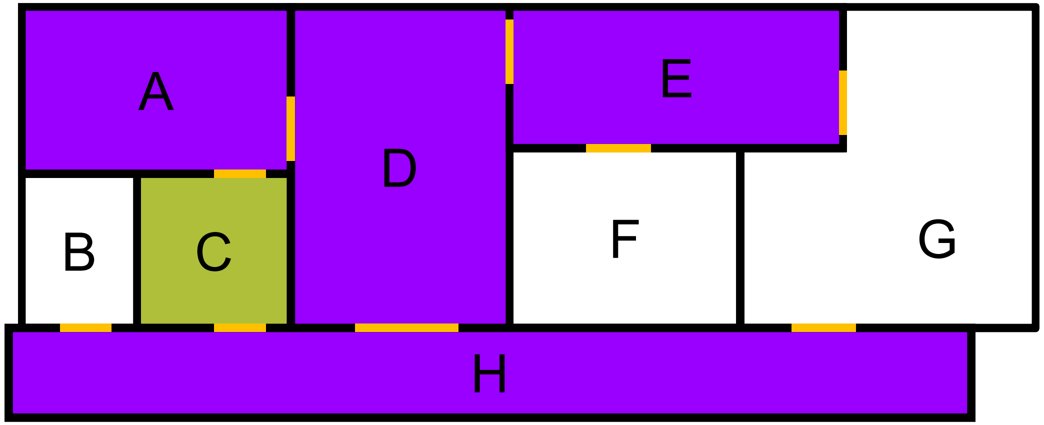


Cells & Portals



Cells & Portals

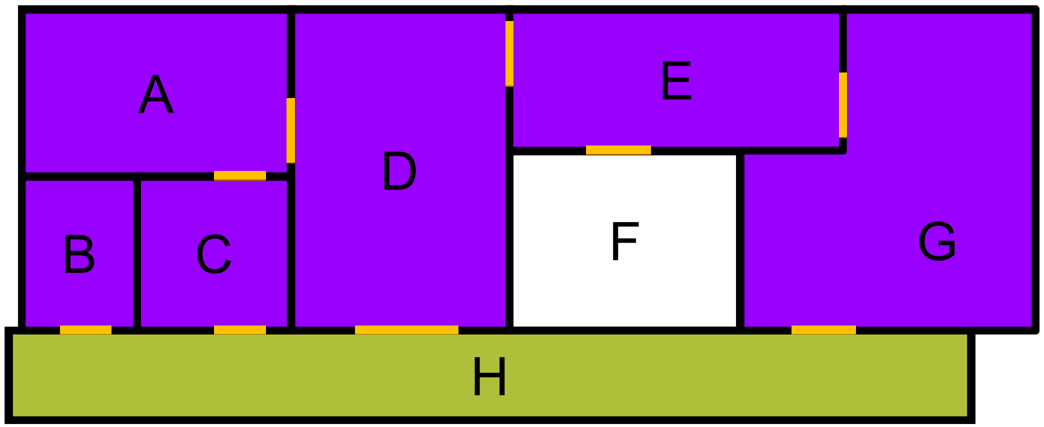
- *View-independent* solution: find all cells a particular cell could *possibly* see:



C can *only* see A, D, E, and H

Cells & Portals

- *View-independent* solution: find all cells a particular cell could *possibly* see:



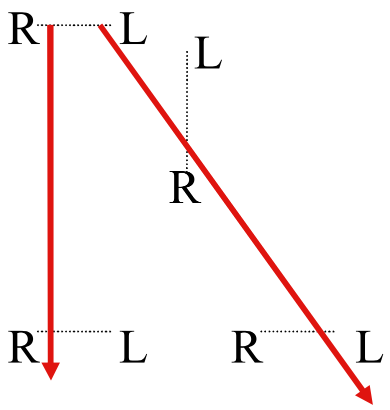
H will *never* see F

Cells & Portals

- Questions:
  - *How can we detect whether a given cell is visible from a given viewpoint?*
  - *How can we detect view-independent visibility between cells?*

Cells & Portals

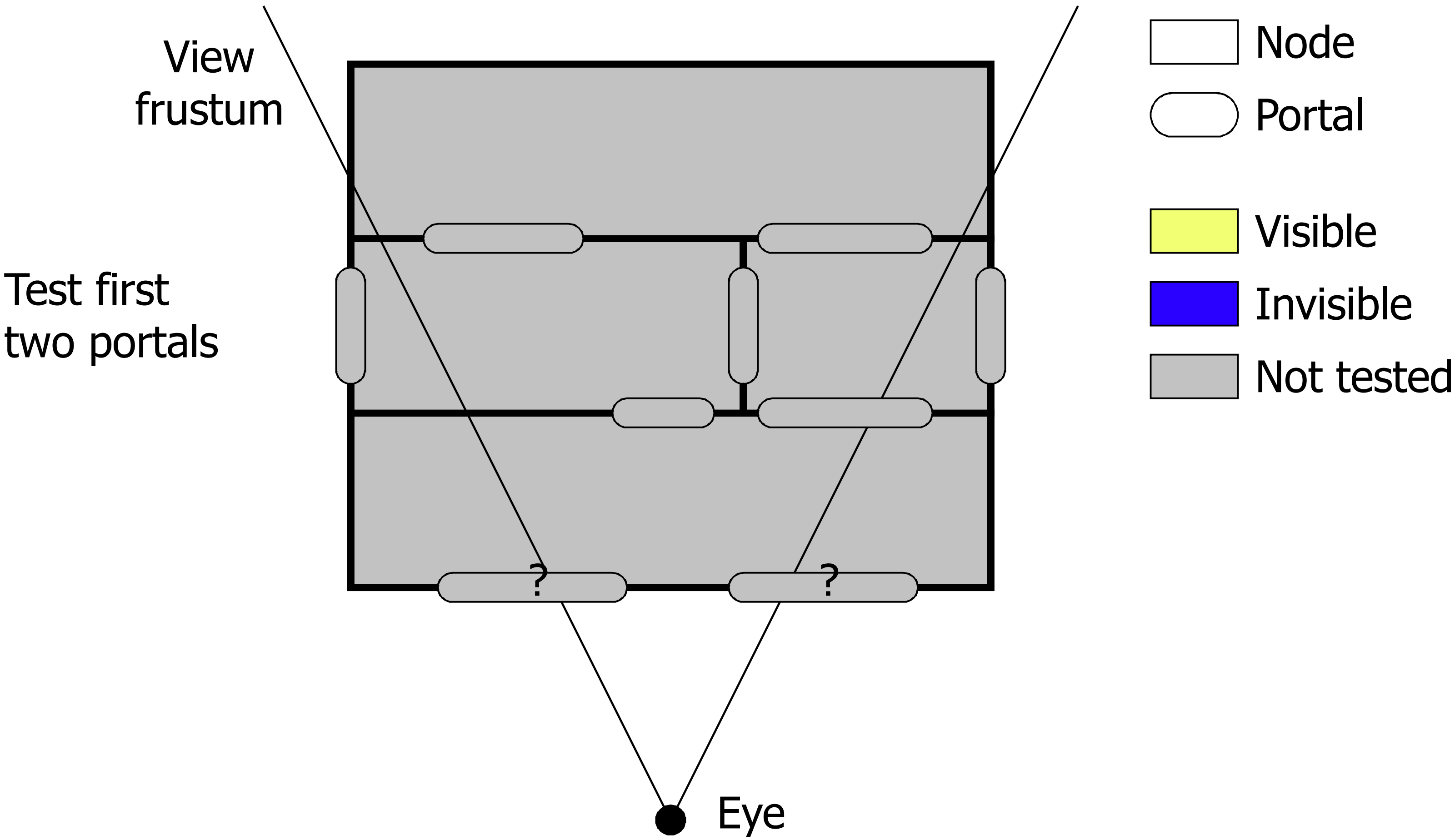
- View-independent method
  - Portal-portal visibility calculated by *line stabbing* using linear program (Teller 1993)
    - Cell A is visible from cell B if there exist a *stabbing line* that originates on a portal of B and reaches a portal of A
    - A *stabbing line* is a line segment intersecting only portals



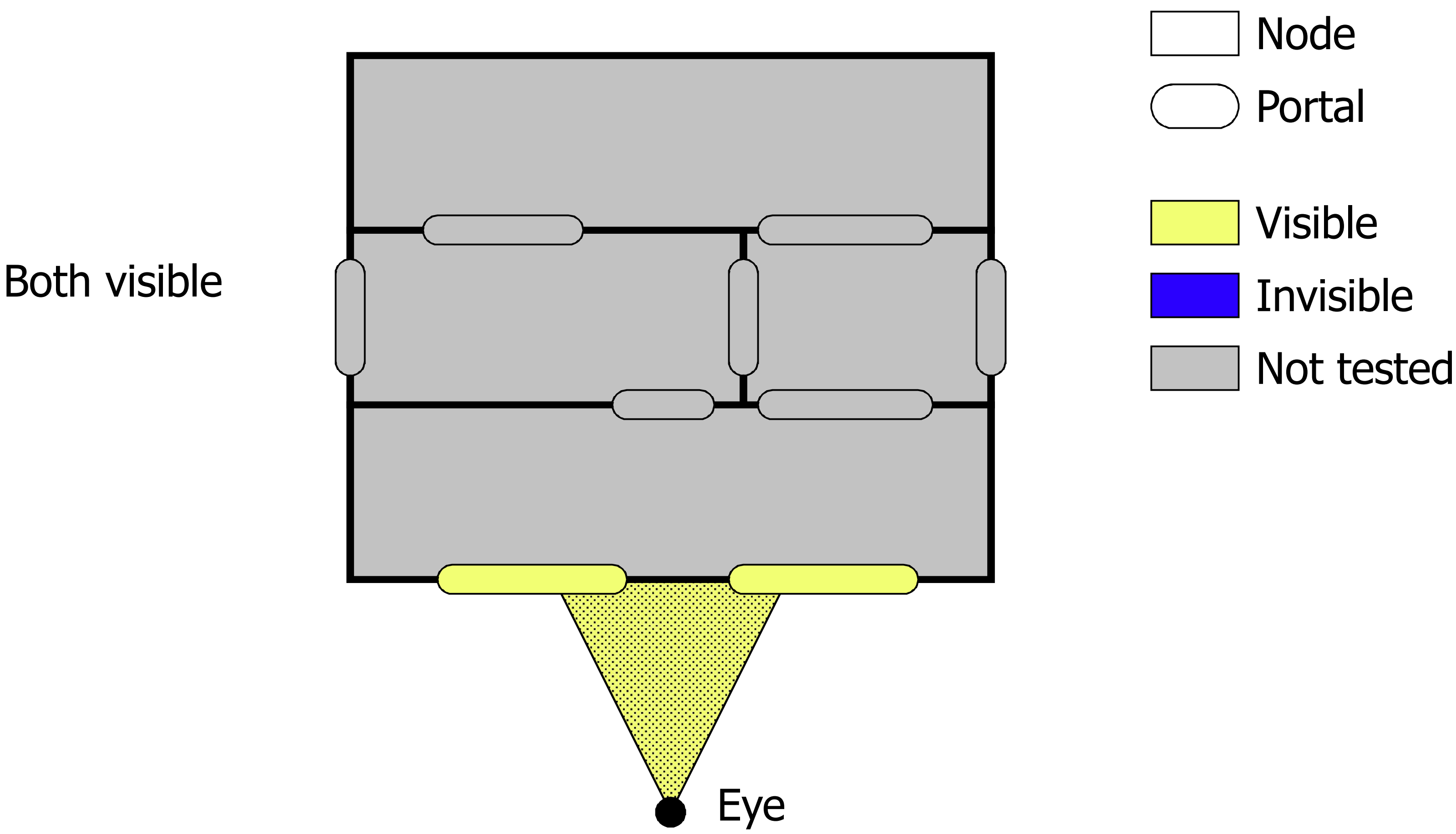
Cells & Portals

- View-dependent method(Luebke, 1995)
  - Nodes joined by portals
    - Usually a polygon, but can be any shape
  - See if any portal of node is visible
  - If so, draw geometry in node
  - See if portals to other nodes are visible
    - Check only against visible portal shape
    - Common to use screen bounding boxes
  - Recurse to other nodes

View Dependent Cells & Portals

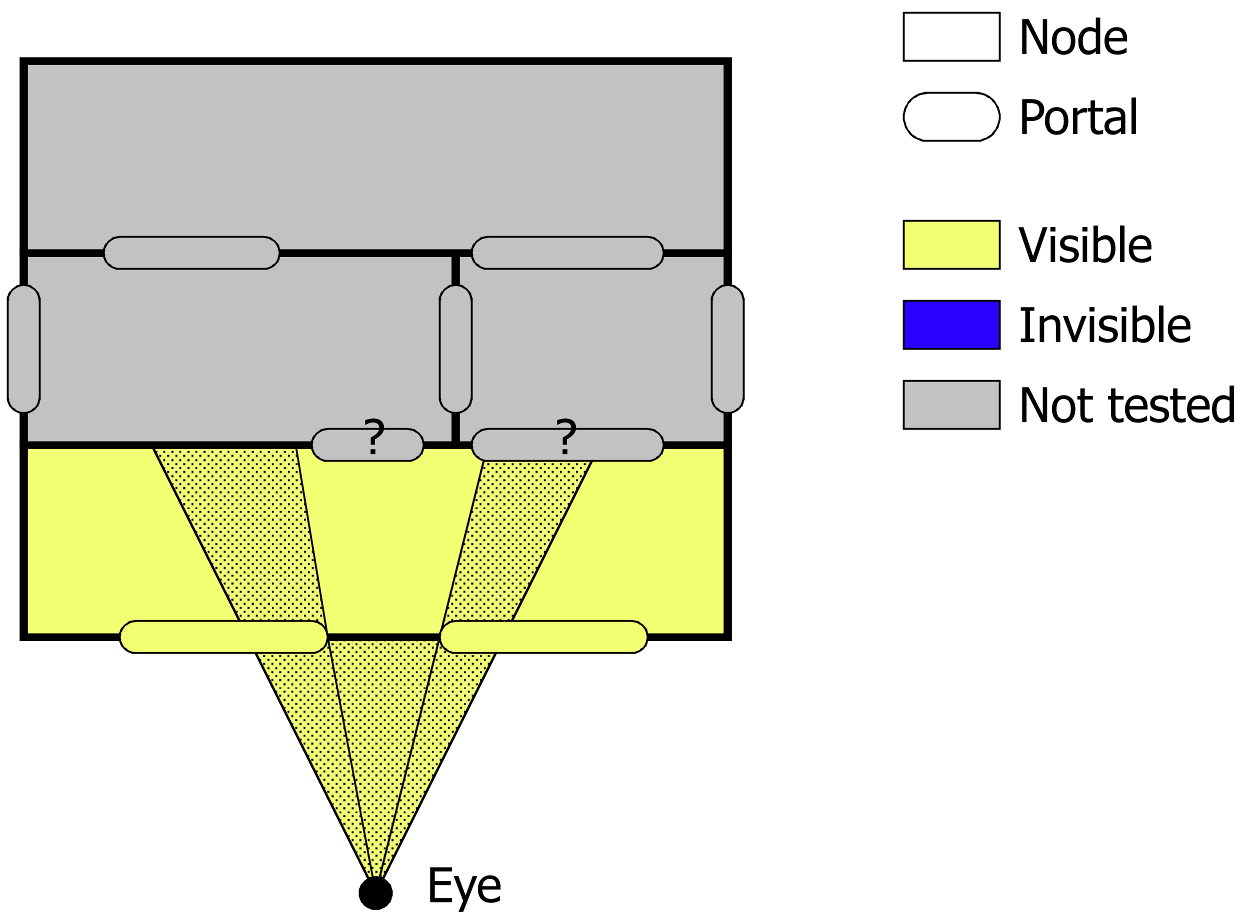


View Dependent Cells & Portals



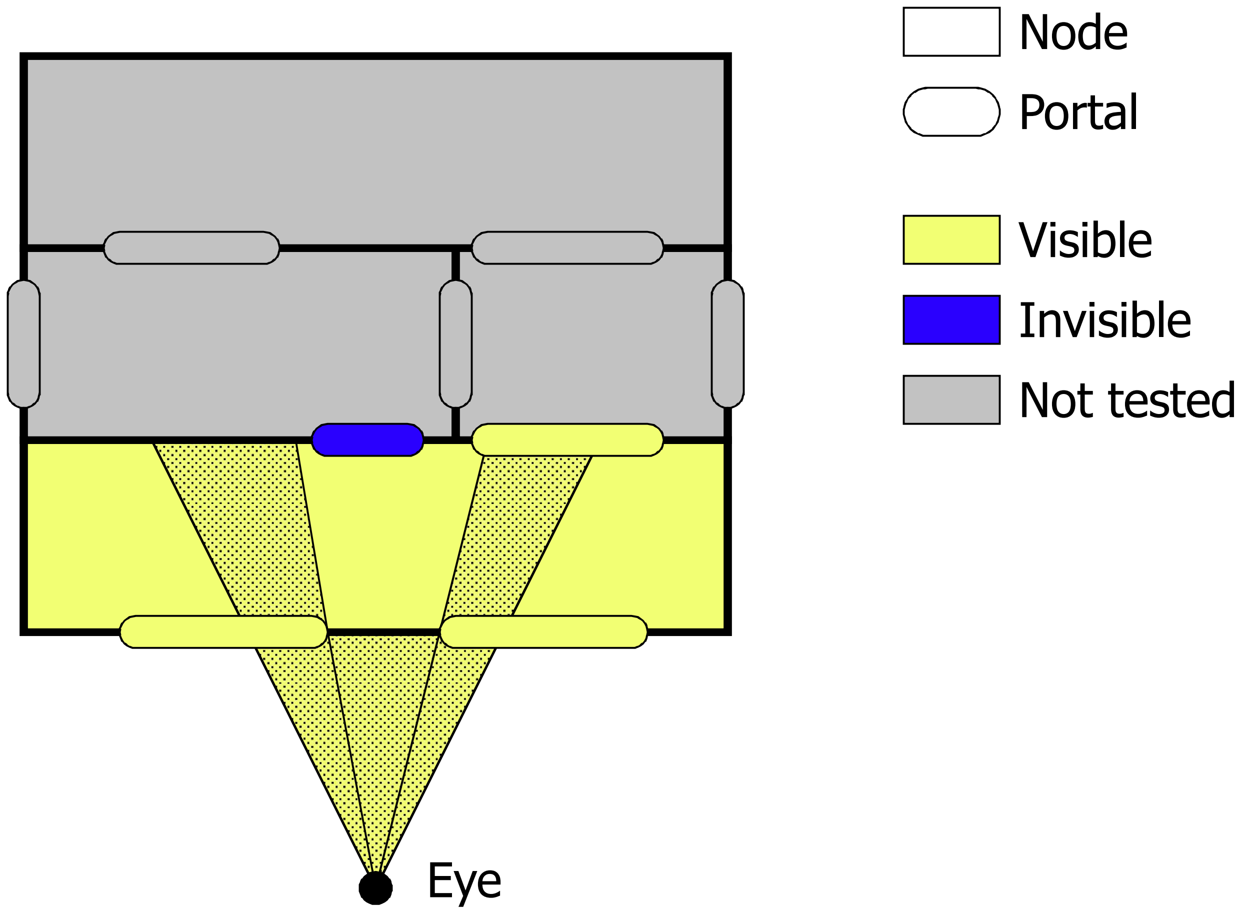
View Dependent Cells & Portals

Mark node visible, test all portals going from node



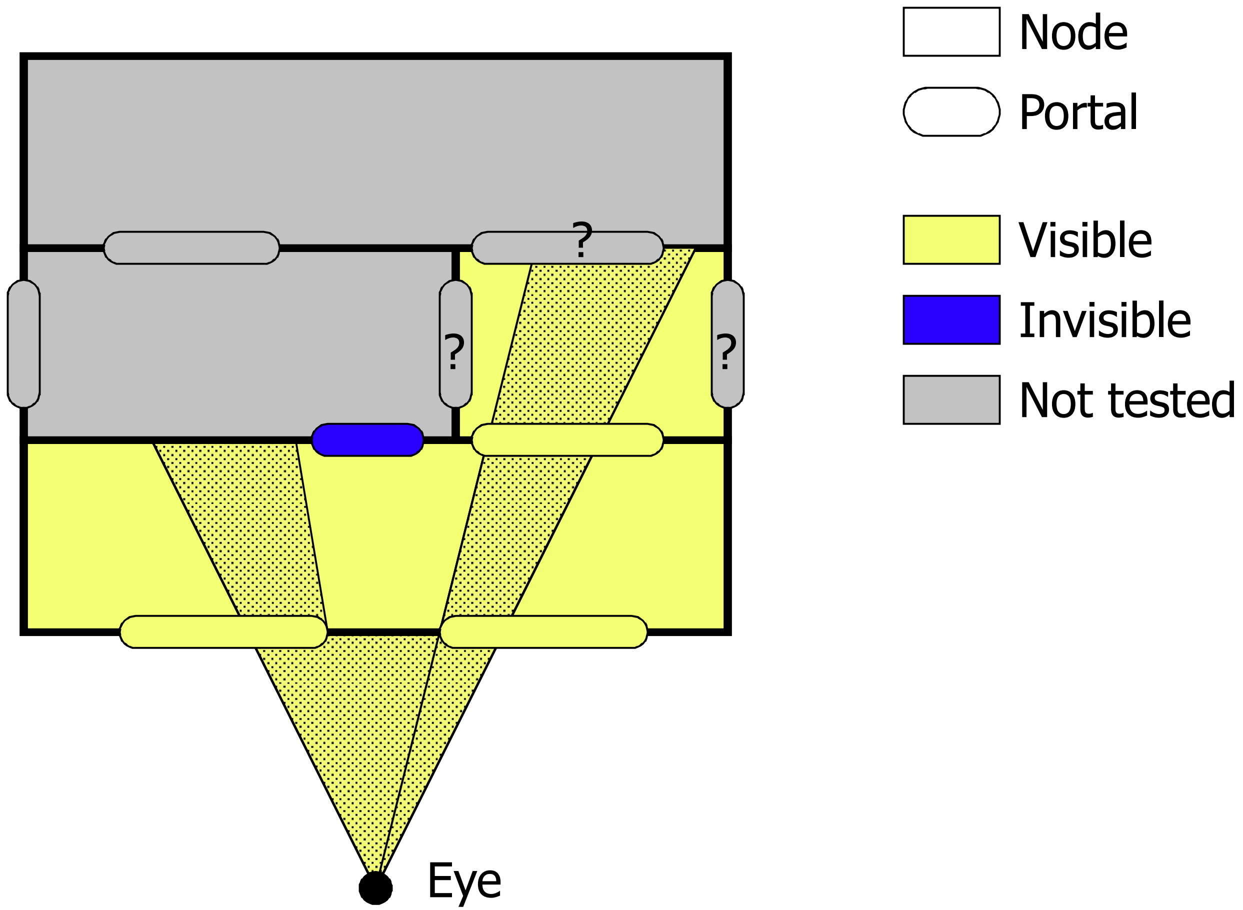
View Dependent Cells & Portals

One portal visible, one invisible



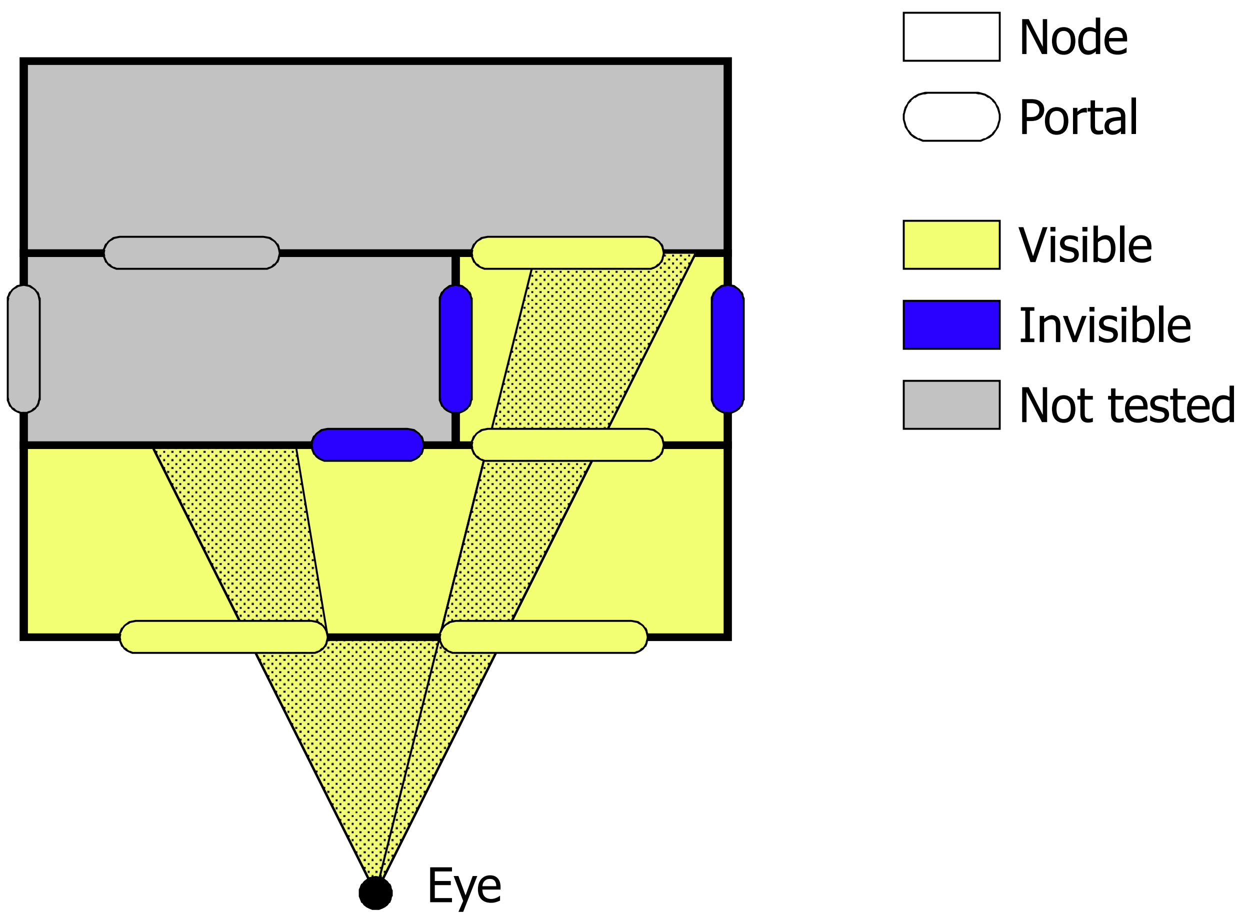
View Dependent Cells & Portals

Mark node as visible, other node not visited at all. Check all portals in visible node



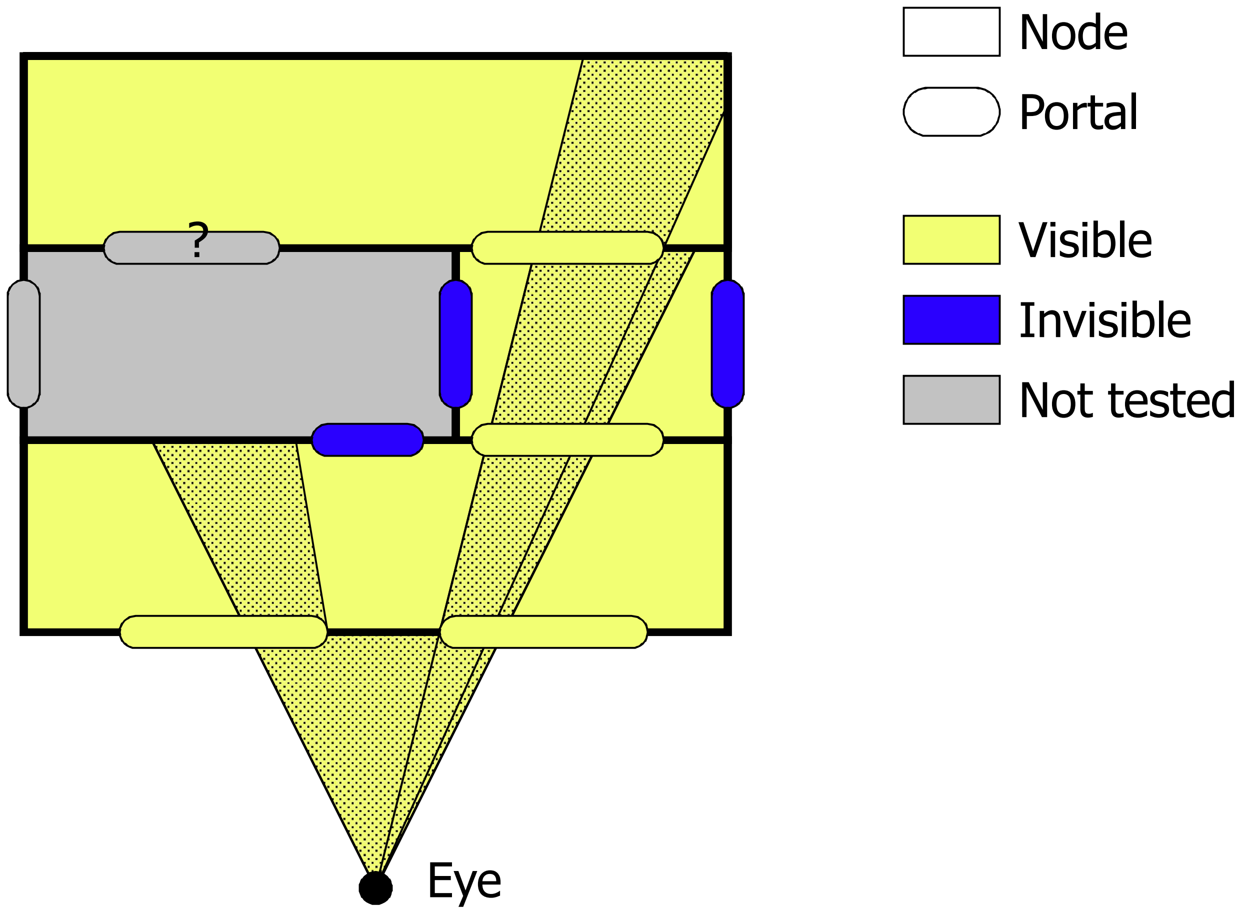
View Dependent Cells & Portals

One visible,
two invisible



View Dependent Cells & Portals

Mark node as visible, check new node's portals



View Dependent Cells & Portals

One portal invisible.
No more visible nodes or portals to check.
Render scene.