CS 563 Advanced Topics in Computer Graphics
Real Time Rendering (Part 1)

Kutty S Banerjee

Real Time Rendering

Broad Classification:

- Geometry Based Rendering
- Image Based Rendering

Have a set of methods lying in between.

- We Start from GBR and gradually move towards IBR incorporating techniques from IBR

GBR

- Scene described using geometric objects
  - How? Using CAD tools, solid modellers..
- Geometry sampled and discretized
- Stored internally as triangles..(tessellation), quads
  - Contain light, normal coordinates
- With information ---- **simulate** the world...why?
- Light equations, Gouraud, Phong, Phong Blinn physics that recreates world lighting using equations
- Complexity proportional to scene complexity

GBR moves towards IBR

- Consider Image Based Techniques
  - Textures
  - Environment Mapping
  - Bump Mapping
  - Image Warping
- Point Based Rendering & Image Based Rendering in second half of talk!!

Textures

- Moving from Pure Geometric Modelling towards IBR
- Instead of modelling and rendering, use textures for color, roughness, reflection, shadows!!
- Vast Topic... cover ideas mostly!!

Cover

- **Texture Mapping**
- **Texture Filtering**
- **Textures in OpenGL**
- **Environment Mapping**
- **Bump Mapping**

Not Cover

- **Rendering of Textures**
- **Interpolation Techniques**

http://www.kenmusgrave.com/pleiades2.jpg

Texture Mapping

- Textures ->1D, 2D, do not cover 3D
- 2D Texture, bitmap , each point **texel**
- **Texture Mapping=**

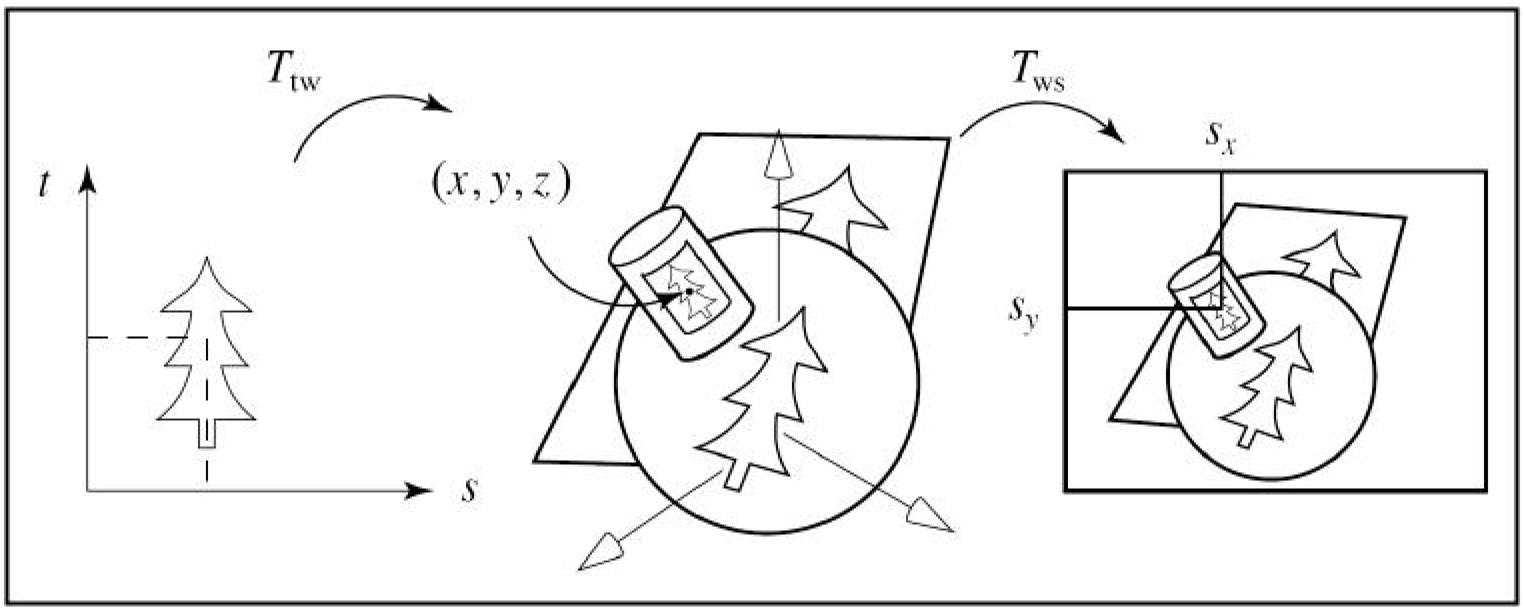
  Journey from Texel(s) **to Pixel**(s)
- **Screen Scanning:**

  For every pixel locate a texel. Most Common!!
- **Texture Scanning:**

  For every texel locate a pixel.

Mapping

- Ttw and Tws are the transforms
- Their inverse transforms can be used as well!!



[1]

Mapping Cases

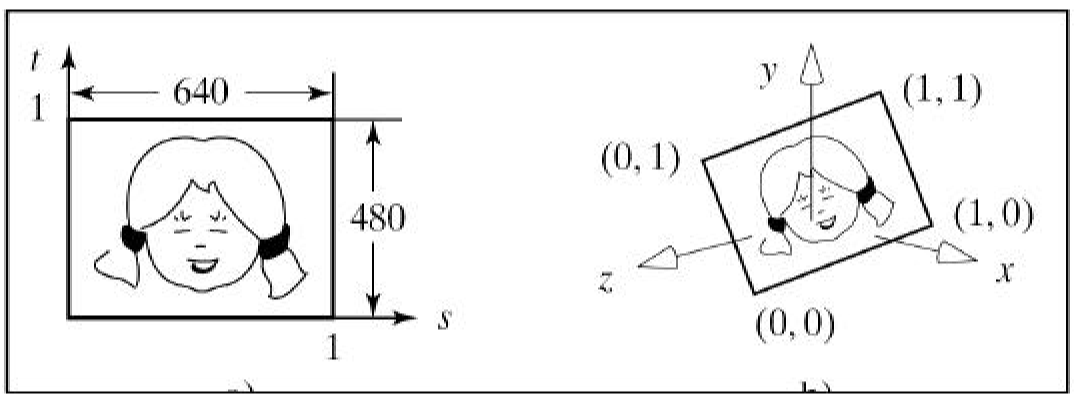
Affine Linear Mapping

We deal with following Mapping Cases:

1. 2D texture to polygon surface (2D)
2. 2D texture to curved surfaces using meshes.
   - Cylinder
   - Cylinder like
   - Sphere
   - Sphere like

To Planar Surface

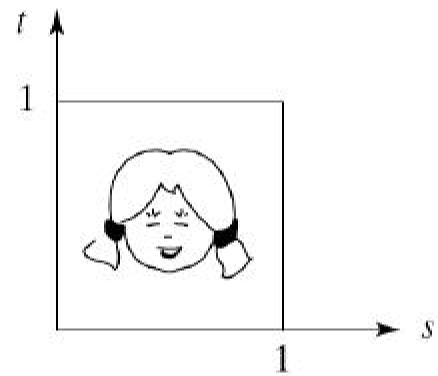
- Affine mapping setup (#vertices same)
- Affine -> Equal ranges in texel space and pixel space
- Linear Transformations
  - Translation, rotation, scaling allowed

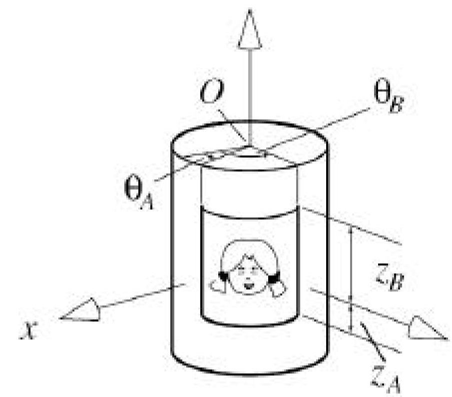


[1]

Curved Surface - Cylinder

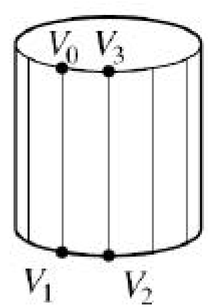
- Cylinder modelled using mesh (quad faces)
- **Patching** advantage -> can use *surface parameters come for free*

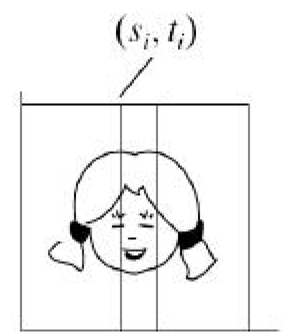




$$s = (\theta - \theta_a)/(\theta_b - \theta_a)$$

$$t = (z - z_a)/(z_b - z_a)$$

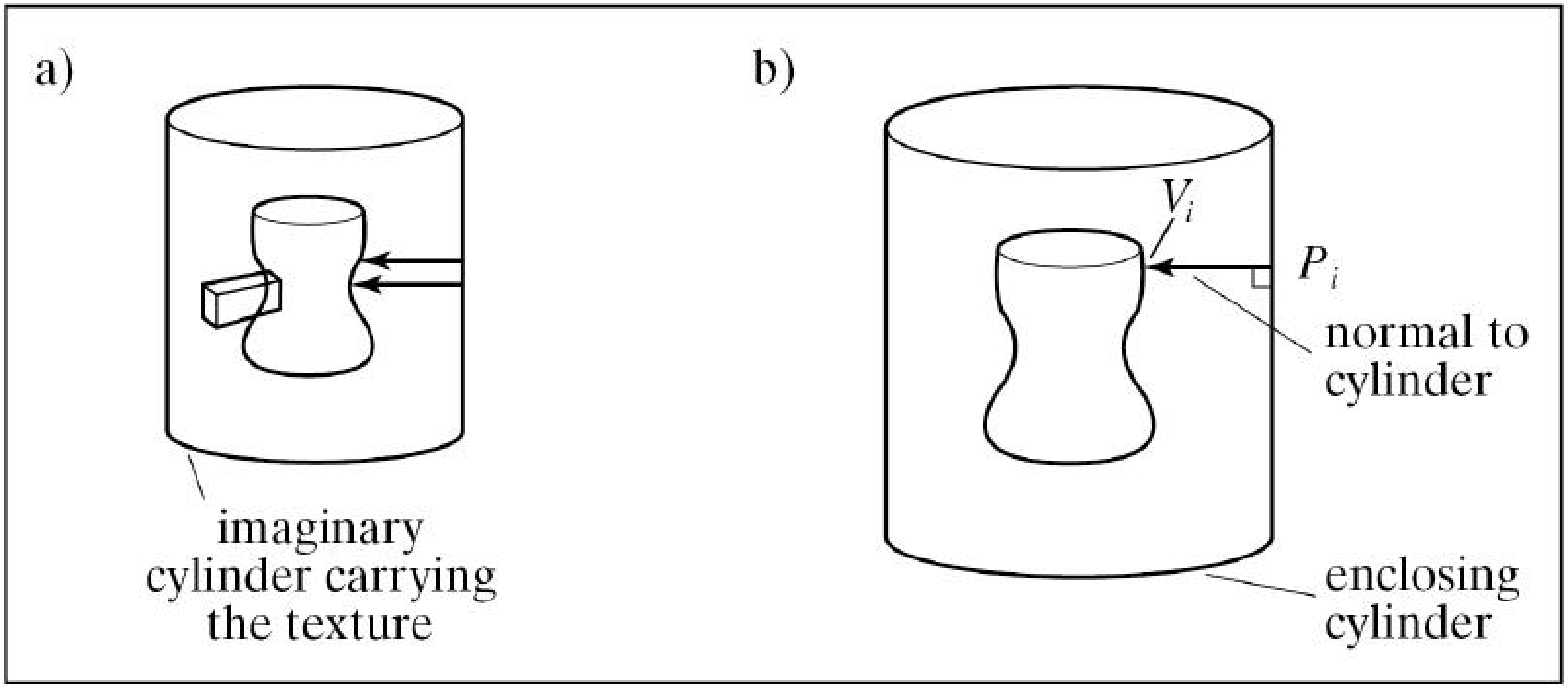




[1]

Cylinder Like

- How about a chess pawn?

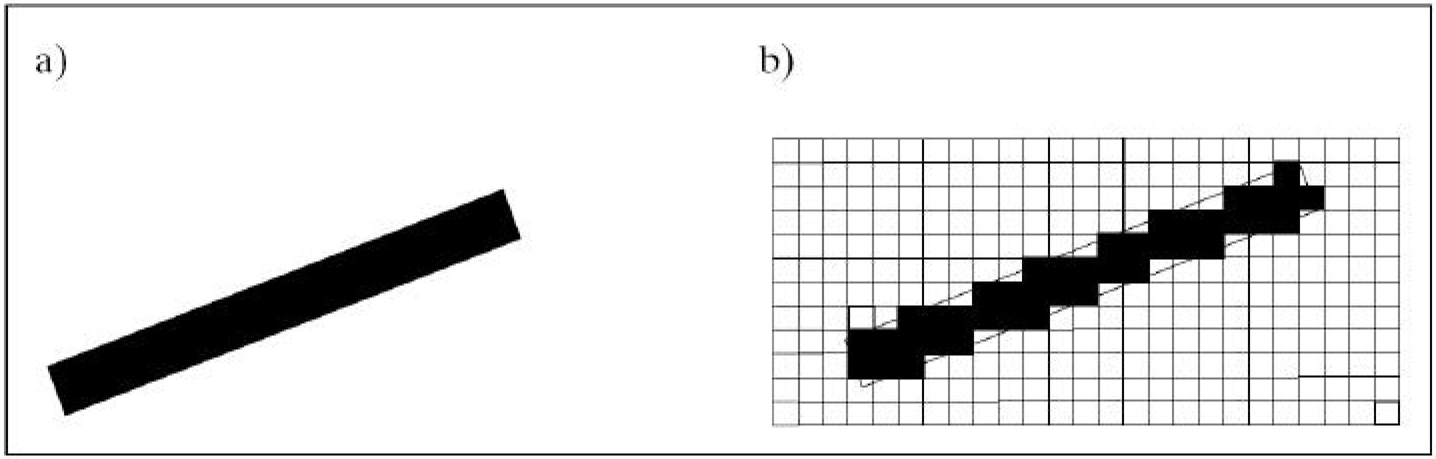


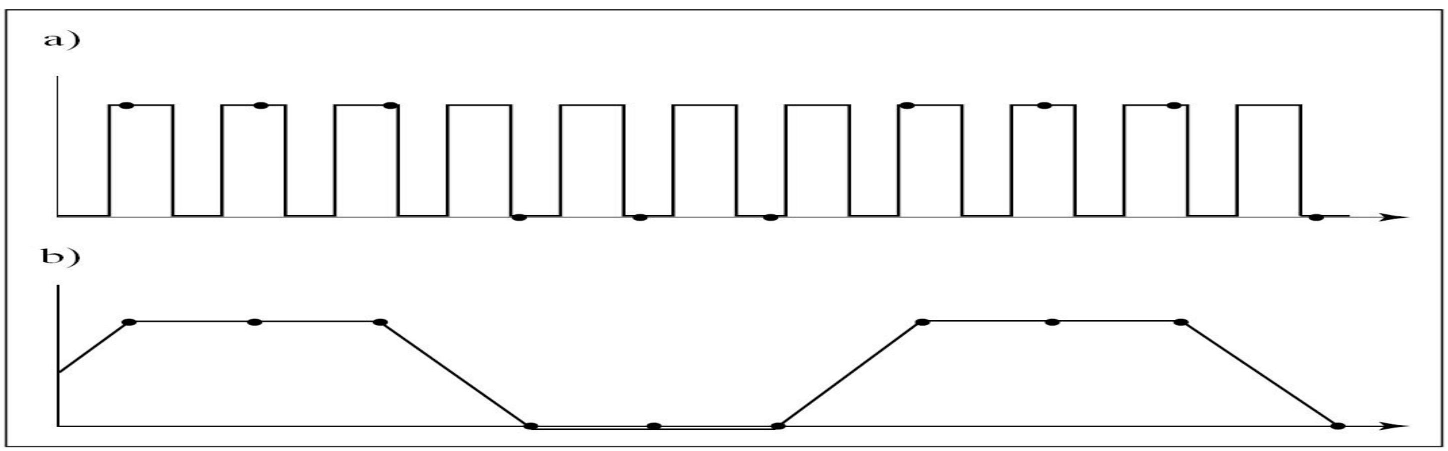
[1]

Texture Filter

Aliasing-Concept

- Sampling high-frequency signal at low-frequency
- Solution? Sample faster!!
- Screen resolution finite!! Can you increase it?



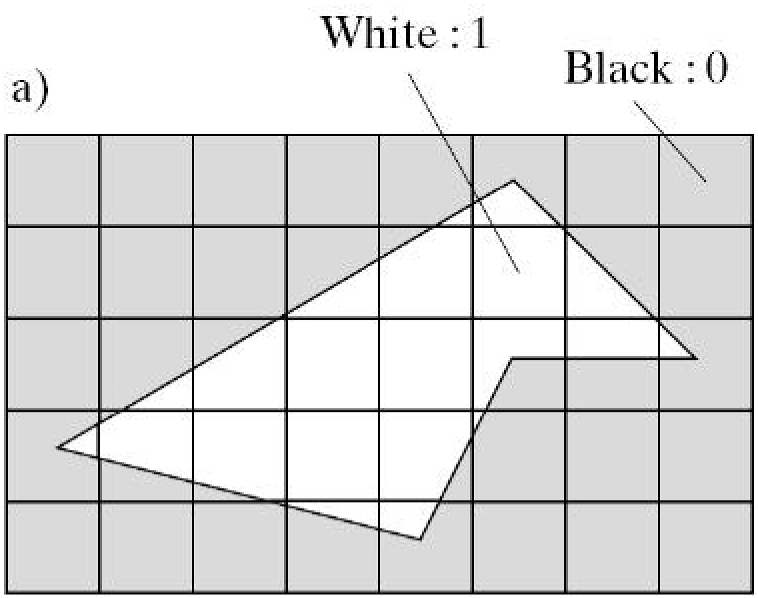


[1]

Pre Filtering

- *Look inside a pixel.*
- *Search for pixel coverage.*

a)



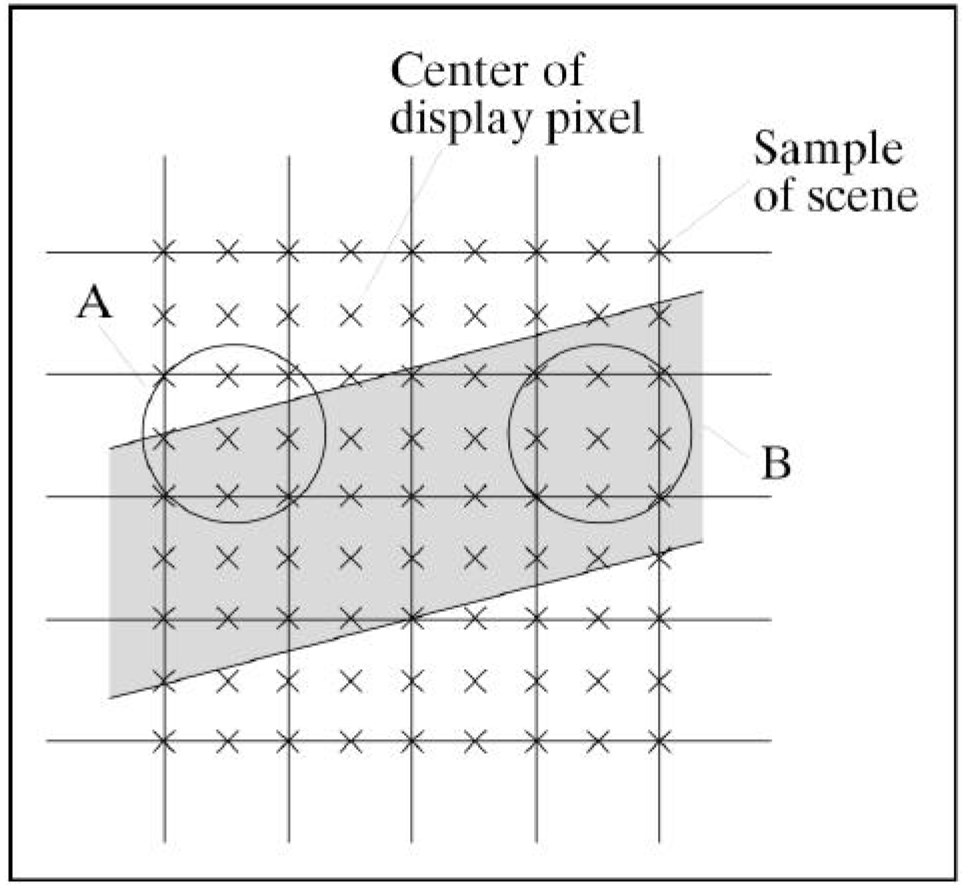
b)

0	0	0	0	1	6	0	0
0	0	0	6	13	15	8	0
0	3	11	15	15	9	7	3
3	11	14	15	12	2	0	0
0	0	1	6	5	0	0	0

[1]

High Sampling

- Increase Sampling Rate.
- So 1 pixel really made up of `n' fragments.
- Consider the color of these fragments
- Mere **Average** or **Weighted Sum**!!

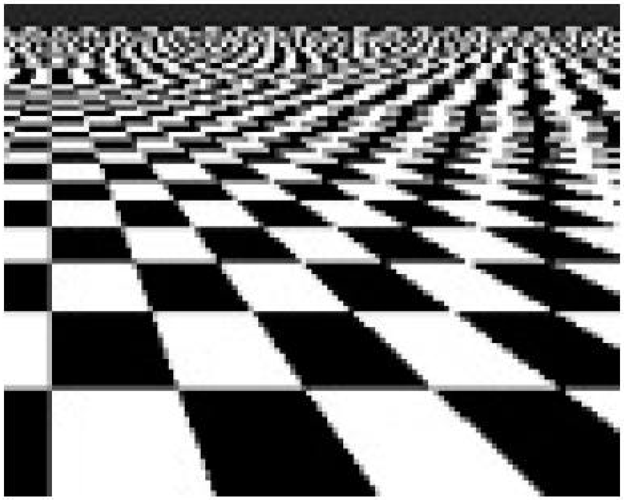


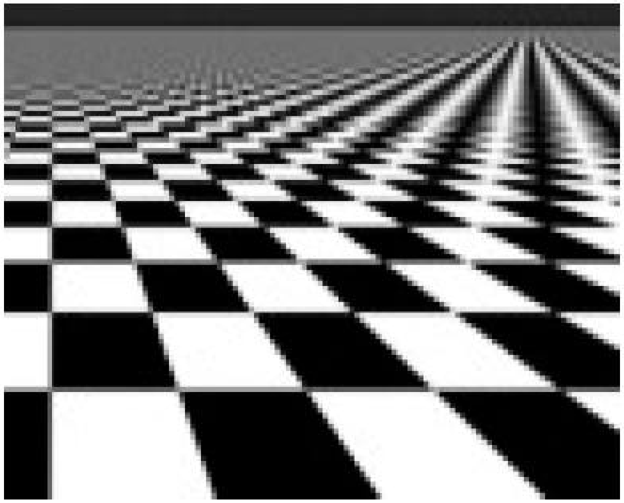
[1]

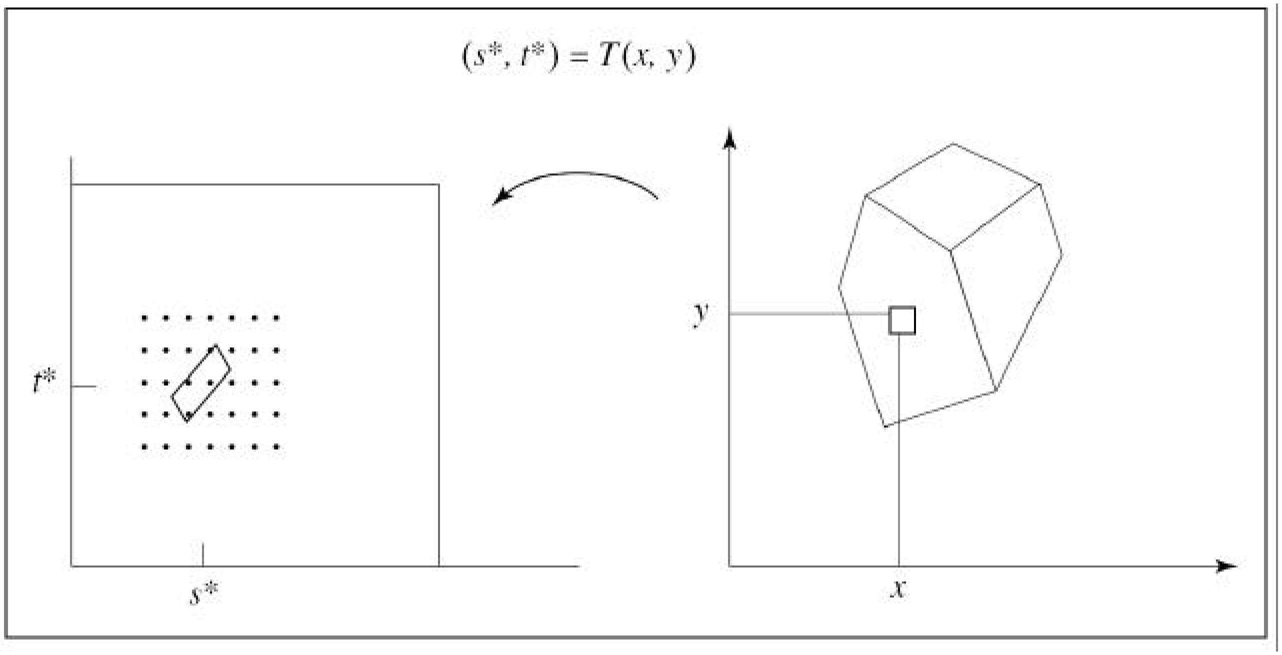
Anti Aliasing in Textures

- ahaa back to textures again……!!
- Screen Pixels not points, have area. Live with it…..
- A pixel point maps to texel.
- But a pixel area maps to what?…..
- A “***set of texels***” of course….
- **Root cause of aliasing problem in textures**

Anti Aliasing Textures







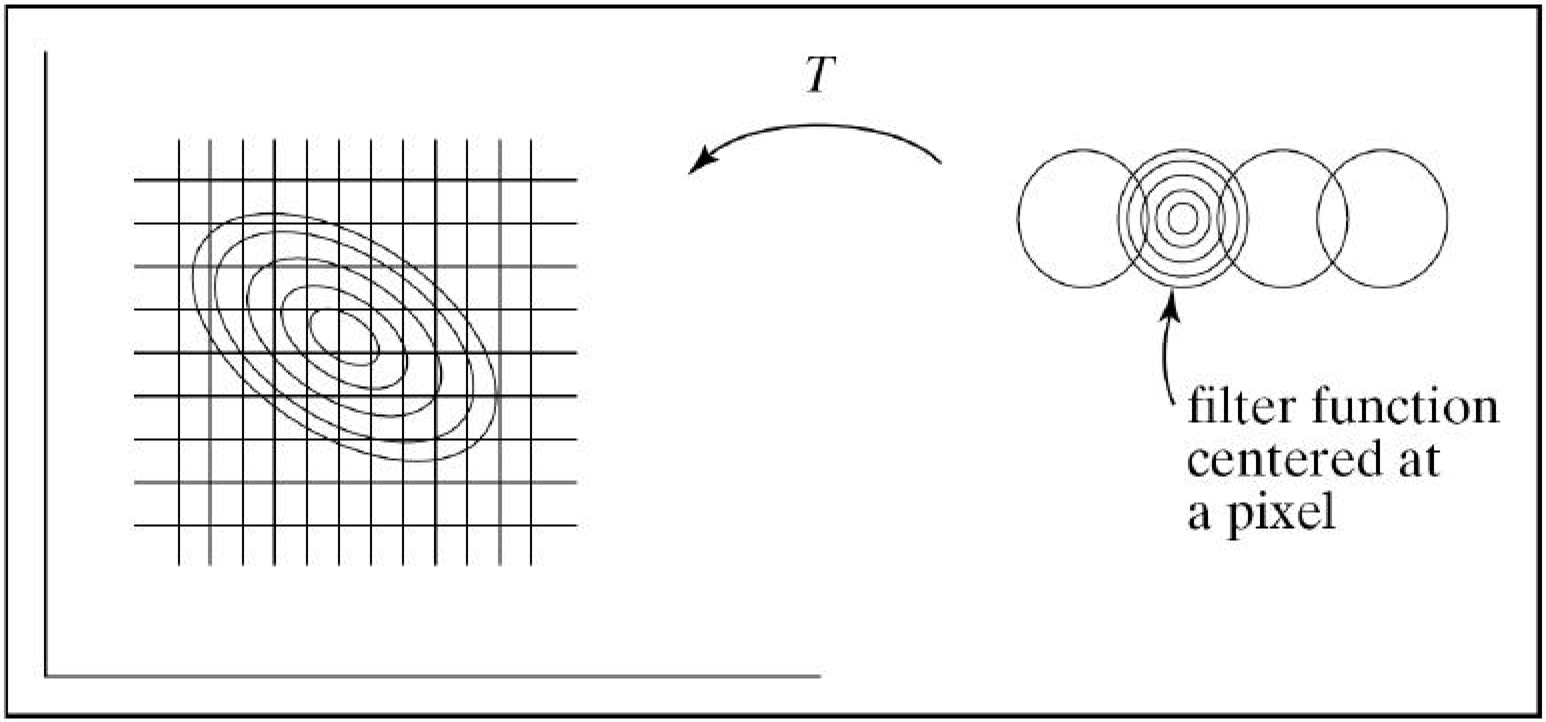
[1]

Filtering Problem

- The central idea behind Filtering
  - *Map* a **pixel** to "**set of texels**"
- How do we do it?
- Elliptical Weighted Average [Heckbert]
- Stochastic Sampling

Elliptical Weighted Average

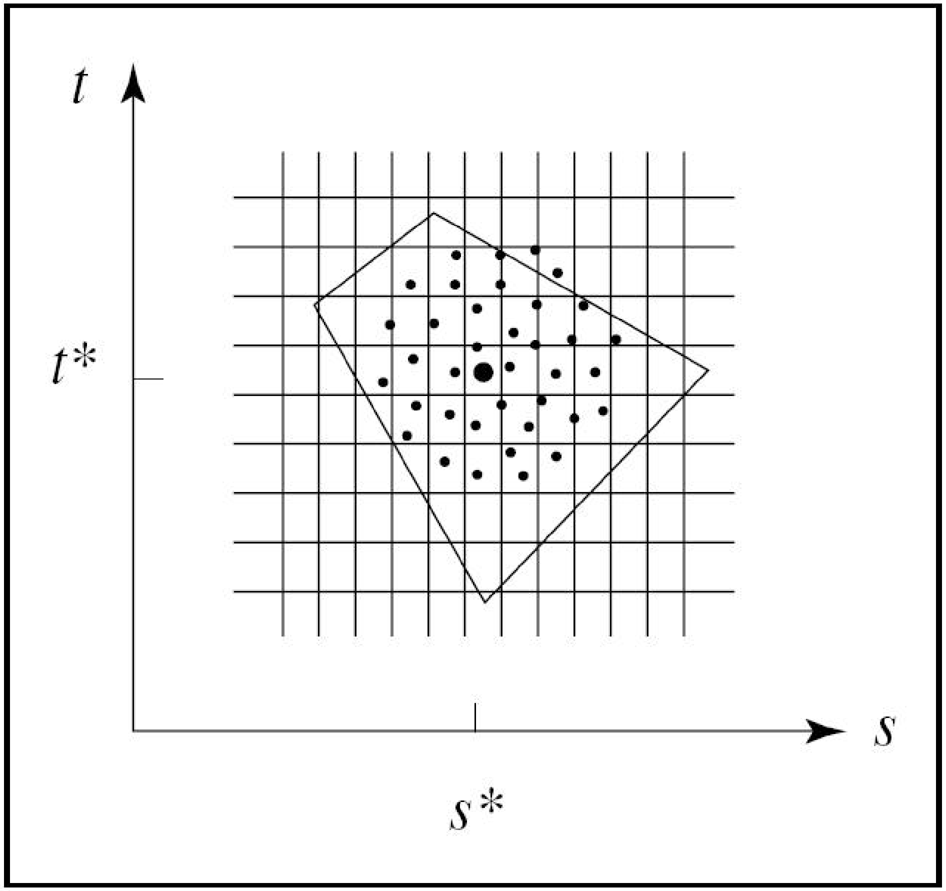
- Every pixel associated with a symmetric filter function
- Generates a circle around the pixel
- Maybe different for each pixel.
- Therefore LUT
- Circle mapped to texel space -> ellipse
- All texels inside ellipse “**average**” or “**weighted sum**”



[1]

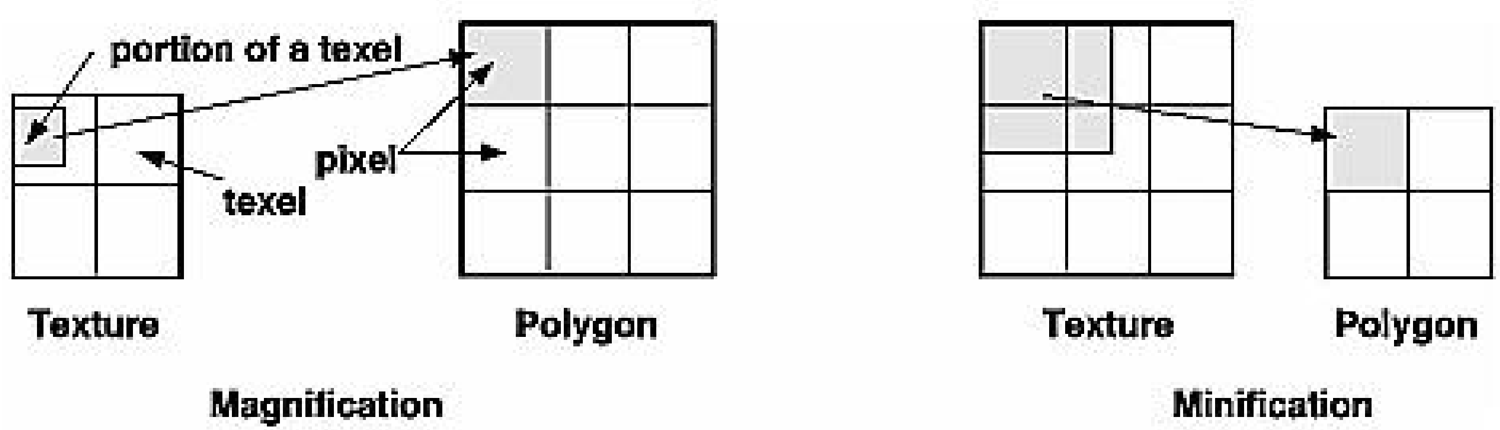
Stochastic Sampling

- Locate texel for the pixel
- Sample surrouding texels using a random function.



[1]

Filtering In OpenGL



- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MAG_FILTER, GL_NEAREST);
- glTexParameteri(GL_TEXTURE_2D, GL_TEXTURE_MIN_FILTER, GL_NEAREST);

[6]

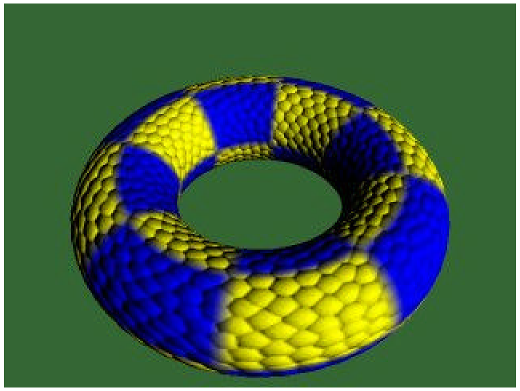
Fancy Textures

Environment Mapping

- So far...only color lifted from textures
- How about reflections?
- One option -> raytracing? Rays bouncing and killing each other.......
- Textures Make it simpler.....eg., Environment mapping!!
- Ray strikes surface.
- From surface find reflection vector
- **Map reflection vector to texels . How?** Different algos...
- Use reflection vector r=e-2(n.e)n [**Blinn, Newell**] e= eye vector, n=normal
- Map 'r' to sphere using
  - P=arccos(-rz)
  - O=atan2(ry,rx)
- Convert P,O to texels (u,v) by normalizing
- **Texture covers sphere surrounding the reflection point**
- **Disadvantage: Need per pixel normal, lighting info.**

Bump Mapping

- Instead of color from texture, use normal stored in texture
- In lighting equation, add this normal with existing normal
- Modelling creases, wrinkles complex, texturing much faster



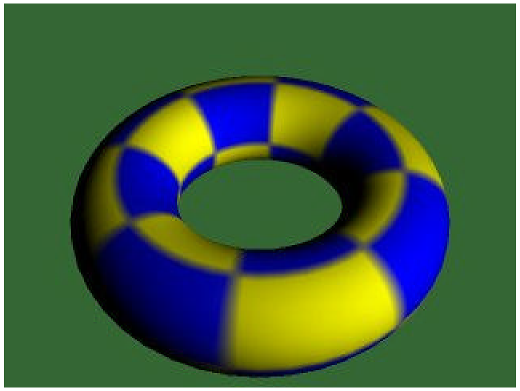


Image Warping

- *Frames have objects in common.*
- *Rendering common objects from scratch wasteful!*
- *Store common objects between frames!! How?*

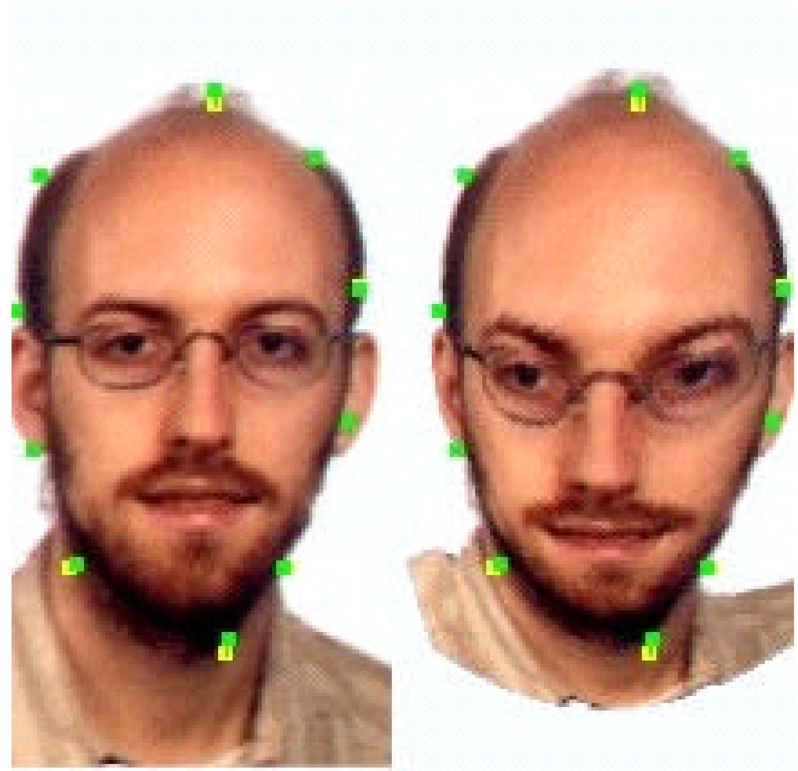
Impostor : Texture image of 3D object on **planar transparent polygon**

Static : Impostors created offline

- Too much memory

Dynamic: Impostors created in real-time

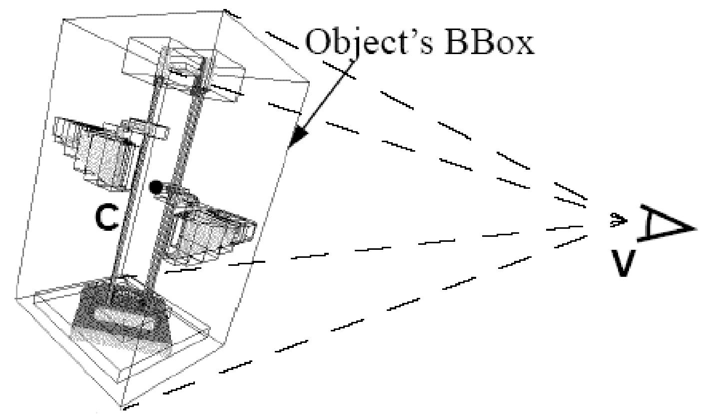
- Processing Rate higher!
- Lower Memory Consumption!

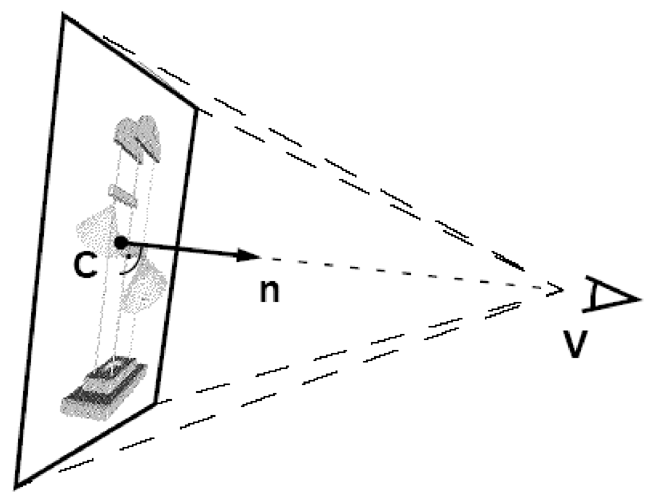


http://www.gris.uni-tuebingen.de/projects/ilo/repository.html

Dynamically Generated Impostors

- ***How are they Created?***
- 3D objects surrounded by Bounding Box
- Journey from FrameBuffer to TextureBuffer!! ☺
- Projn of BBox
- Wrap Smallest Rectange around it.
- 2D Image "view-dependent"





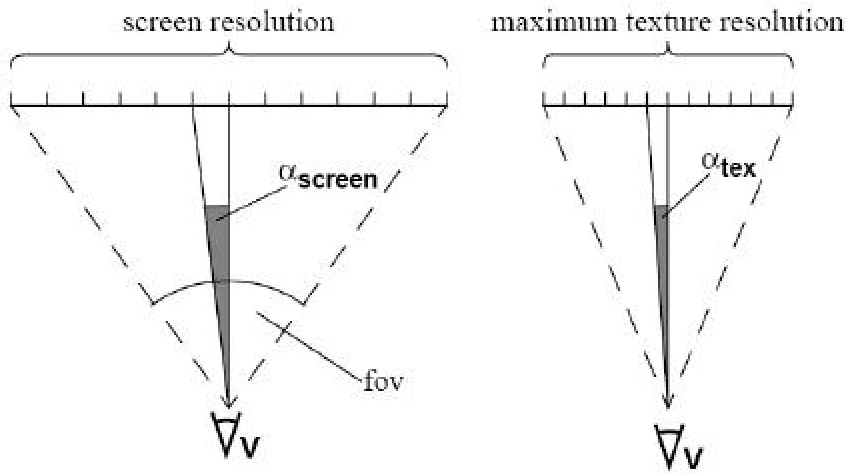
[2]

Congratulations!! You just created an Impostor!!

When are they used?

- Normally what would we think?
  - Further objects impostors rite?
- When *view angle of texel <*
  view angle of pixel
  Use Impostor!!

$$\alpha_{tex} < \alpha_{screen}$$



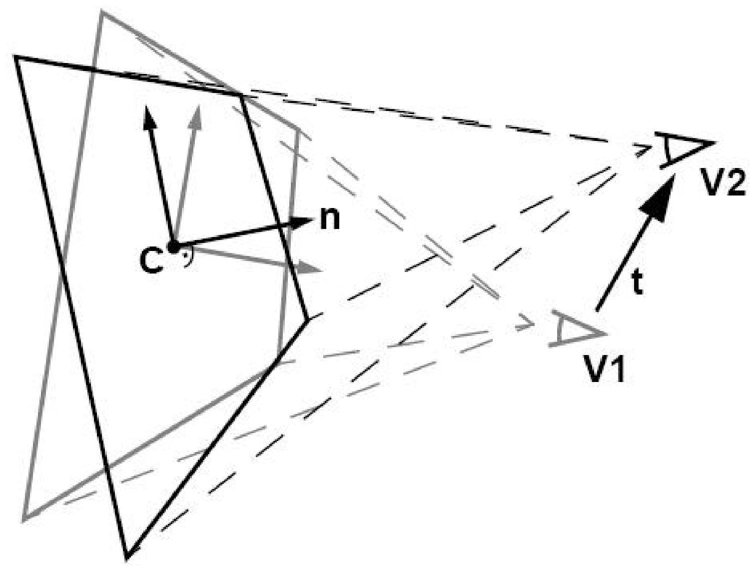
[2]

How steady is the EYE?

- What if **object moves**?
- What if **eye rotates**?
- What if **eye moves towards object**?
- What if **object moves towards eye**?

Object Moves? -> Nopes shouldn’t. Seriously, limitation of this work.

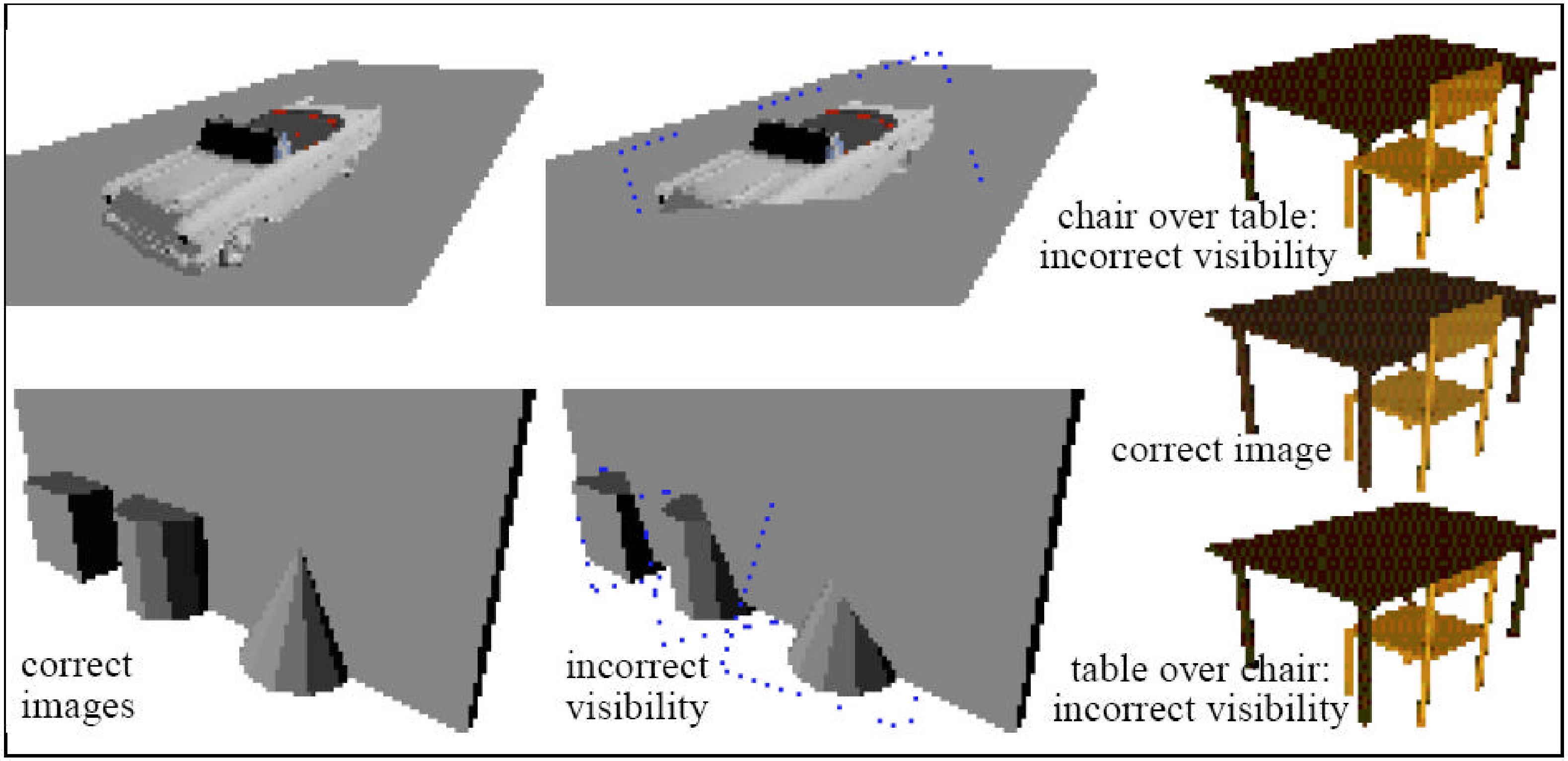
Eye **rotates, translates within minimum range**, 2D affine transformation of Impostor solves problem.



[2]

What about Occlusion?

- Depth Testing?
- Depth Stored per-impostor.
  All texels have same depth!!
- Intersecting objects cause problem!!



[5]

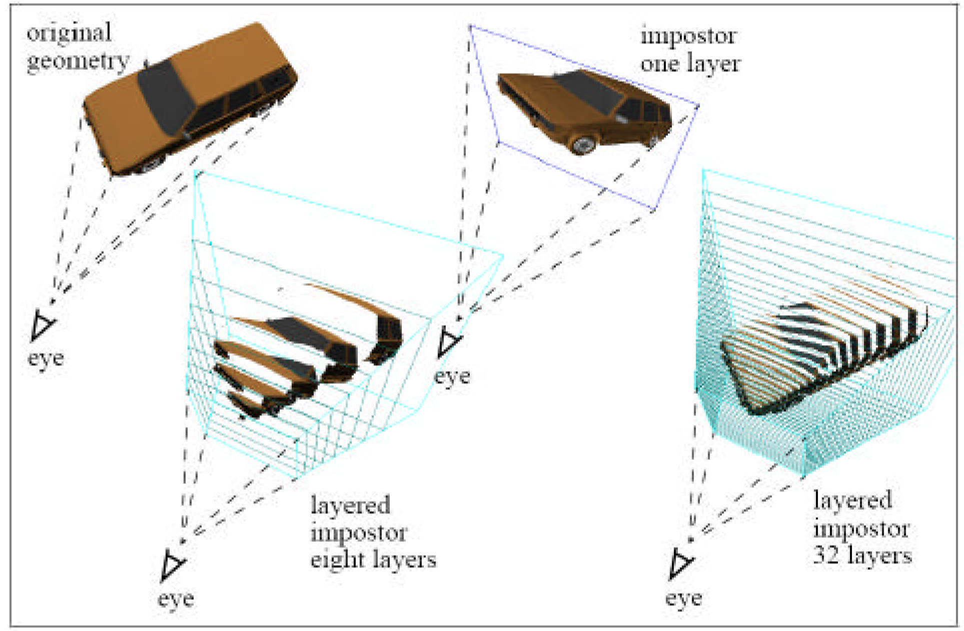
Personal Opinion

- **Nov 95.** Hmmmm.... Quite old!!
- Impostors reduce photorealistic quality of image
- Is video rate available without Impostors modern day h/w?
- Complexity of scene decides.
- Not very clear....

Depth of Impostors' polygon= MaxDepth of any pixel in object's image rite?

MultiLayered Impostors

- Multiple polygonal planes
- Therefore, multiple depth values
- Can reasonably solve object intersection..hmmmmm
  - Maybe with higher #planes, gets better
- With Translation, different layers become distinct!!



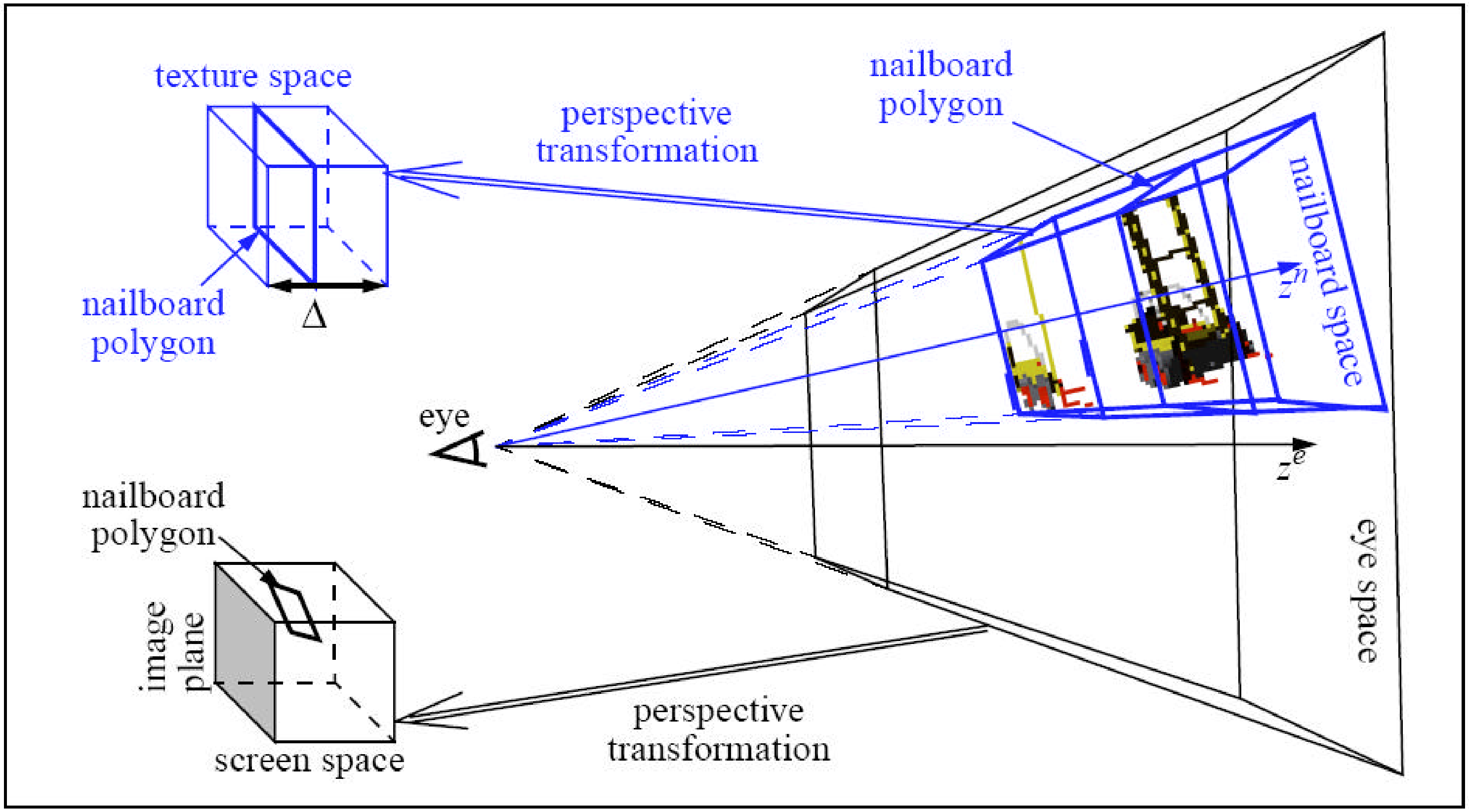
[5]

Nailboards

- Addition onto Impostors
- Stores depth value of each texel
- While copying FB-> TB peeks at **Depth Buffer** too!!
- Since each FB element has corresponding **Depth INFO!!**
- **Texel => (R,G,B,z)**
- Accurate for Object Intersections
- Drawback: Memory Consumption High!!

Nailboards

$$\begin{bmatrix} x^t \\ y^t \\ z^t \\ w^t \end{bmatrix} = \begin{bmatrix} \frac{2n}{r-l} & 0 & \frac{r+l}{r-l} & 0 \\ 0 & \frac{2n}{t-b} & \frac{t+b}{t-b} & 0 \\ 0 & 0 & -\left(\frac{f+n}{f-n}\right) & \frac{-2fn}{f-n} \\ 0 & 0 & -1 & 0 \end{bmatrix} \cdot \begin{bmatrix} x^n \\ y^n \\ z^n \\ 1 \end{bmatrix} \text{ and } X^t = \begin{bmatrix} \frac{x^t}{w^t} \\ \frac{y^t}{w^t} \\ \frac{z^t}{w^t} \end{bmatrix}$$



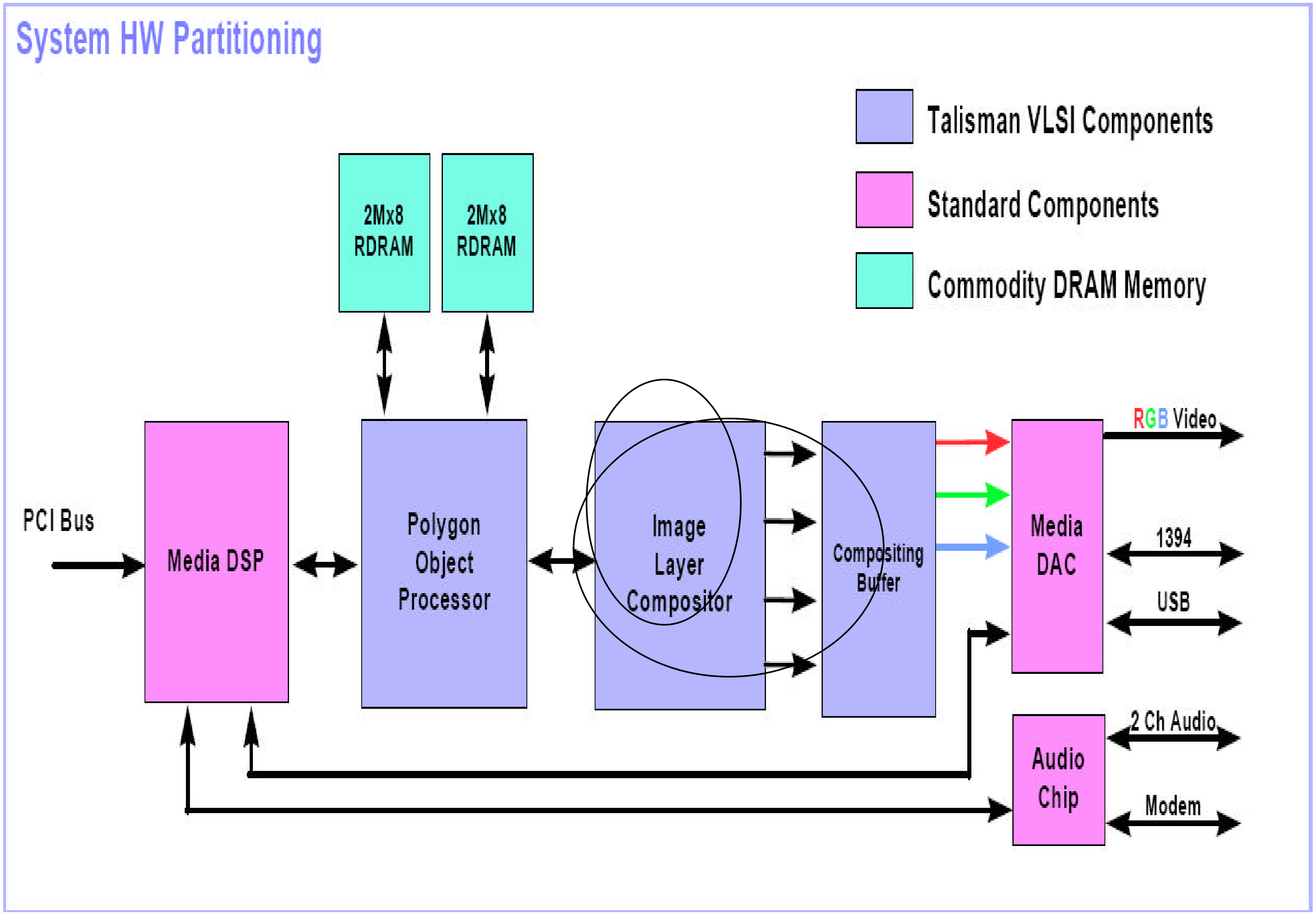
[4]

H/W Support Image Warping

Requirements from h/w?

- **Multiple** Image Layers
- 2D **simulation of 3D Transforms**
- **Real** fast texture memory b/w
- **Sizeable** texture memory
- Geometric , Image **error calc in h/w**

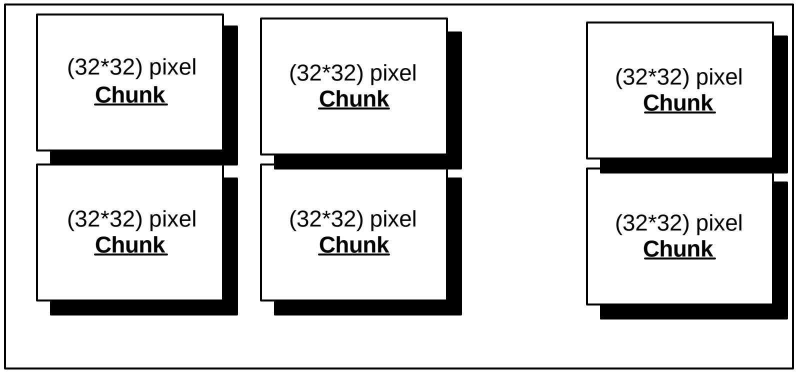
Talisman – Microsoft (1996)

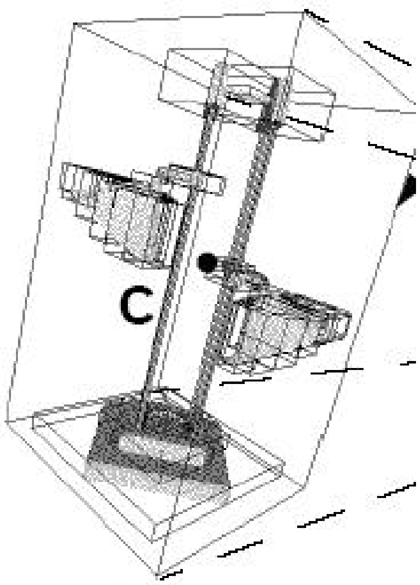


[2]

Composited Imaging

- No *FrameBuffer*
- *Image Layers* with multiple *chunks.*
- Images rendered on Image Layer independently!!
- So *object per Image Layer*

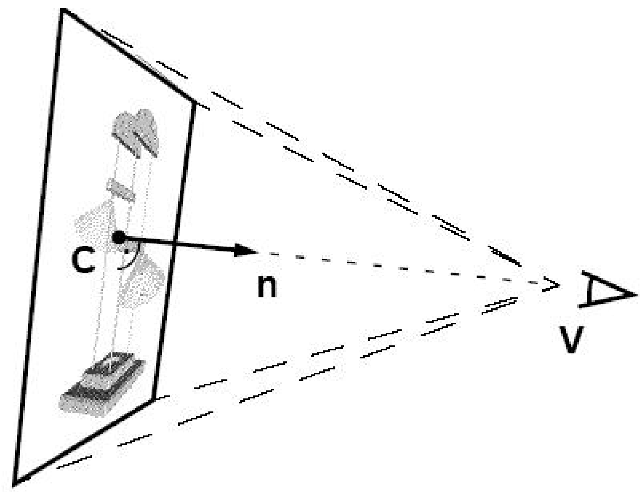




[3]

Simulating 3D Affine Transformations

- Image Layer can 2D transform
- 2D transform to simulate 3D affine transforms
- Less Expensive
- View point rotates
- View point translates (small margin)



[2]

Chunks- Advantages

- Objects sorted (in s/w by programmer) into chunks
  - How? Object level partitioning (voxels)
  - Mapping voxels to chunks
  - Overlapping voxels copied to chunks
- 32*32 => One chunk at a time rendered!!
- Therefore, Z-Buffer how big?
- Texture memory how big?
- Can they both reside on board? Blazing Speed!!
- Objects (Image Layer) prioritizing in S/W

References

1. Computer Graphics using OpenGL- *FS Hill*
2. Talisman: Commodity Realtime 3D Graphics for PC – *Jay Torborg, James T.Kajiya*
3. Dynamically Generated Imposters – *Gernot Schaufler*
4. Per-Object Image Warping With Layered Impostors – *Gernot Schaufler*
5. Nailboards: A Rendering Primitive for Image Caching in Dynamic Scenes – *Gernot Schaufler*
6. OpenGL Programming Guide – *Addison Wesley*