

# **CS 380 - GPU and GPGPU Programming**

## **Lecture 13: GPU Texturing 3**

Markus Hadwiger, KAUST

# Reading Assignment #7 (until Mar. 16)



## Read (required):

- Interpolation for Polygon Texture Mapping and Shading, Paul Heckbert and Henry Moreton

<http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.48.7886>

- MIP-Map Level Selection for Texture Mapping

<http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=765326>

## Read (optional):

- Frame buffer objects extension specification

[http://www.opengl.org/registry/specs/ARB/framebuffer\\_object.txt](http://www.opengl.org/registry/specs/ARB/framebuffer_object.txt)

# Semester Project (Proposal until March 23!)



- Choosing your own topic encouraged!  
(we can also suggest some topics)
  - Pick something that you think is really cool!
  - Can be completely graphics or completely computation, or both combined
  - Can be built on CS380 frameworks, NVIDIA OpenGL SDK, or CUDA SDK
- **Write short (1-2 pages) project proposal until March 23**
  - Send email or talk briefly with Peter before (!) you start writing to confirm that your plan will be a suitable topic
  - Submit with your programming assignment 3 solution (it's the same deadline)
- Submit semester project with report (deadline: May 11)
- Present semester project (final exams week May 17 - 21)

# Semester Project Ideas (1)



## Some hints for topics

- Procedural shading with noise + marble etc. (GPU Gems 2, chapter 26)
- Procedural shading with noise + bump mapping (GPU Gems 2, chapter 26)
- Subdivision surfaces (GPU Gems 2, chapter 7)
- Ambient occlusion, screen space ambient occlusion
- Shadow mapping, hard shadows, soft shadows
- Deferred shading
- Particle system rendering + CUDA particle sort
- Advanced image filters: fast bilateral filtering, Gaussian kD trees
- PDE solvers (e.g., anisotropic diffusion filtering, 2D level set segmentation, 2D fluid flow)

# Semester Project Ideas (2)



## Some hints for topics

- Distance field computation (GPU Gems 3, chapter 34)
- Livewire (“intelligent scissors“) in CUDA
- Comparison of parallel sorting algorithms
- Parallel computation of summed area table / multires pyramid, ... use in rendering
- Linear systems solvers, matrix factorization (Cholesky, ...); with/without CUBLAS
- CUDA + matlab
- Fractals (Sierpinski, Koch, ...)
- Image compression

- Basically, everything done in hardware
- `gluBuild2DMipmaps()` generates MIPmaps
- Set parameters in `glTexParameter()`
  - `GL_TEXTURE_MAG_FILTER: GL_NEAREST, GL_LINEAR, ...`
  - `GL_TEXTURE_MIN_FILTER: GL_LINEAR_MIPMAP_NEAREST`
- Anisotropic filtering is an extension:
  - `GL_EXT_texture_filter_anisotropic`
  - Number of samples can be varied (4x,8x,16x)
    - Vendor specific support and extensions



- Specified manually (`glMultiTexCoord()`)
- Using classical OpenGL texture coordinate generation
  - Linear: from object or eye space vertex coords
  - Special texturing modes (env-maps)
  - Can be further modified with texture matrix
    - E.g., to add texture animation
  - Can use 3rd or 4th texture coordinate for projective texturing!
- Shader allows complex texture lookups!



# Texture Mapping

---

2D (3D) Texture Space

| Texture Transformation

2D Object Parameters

| Parameterization

3D Object Space

| Model Transformation

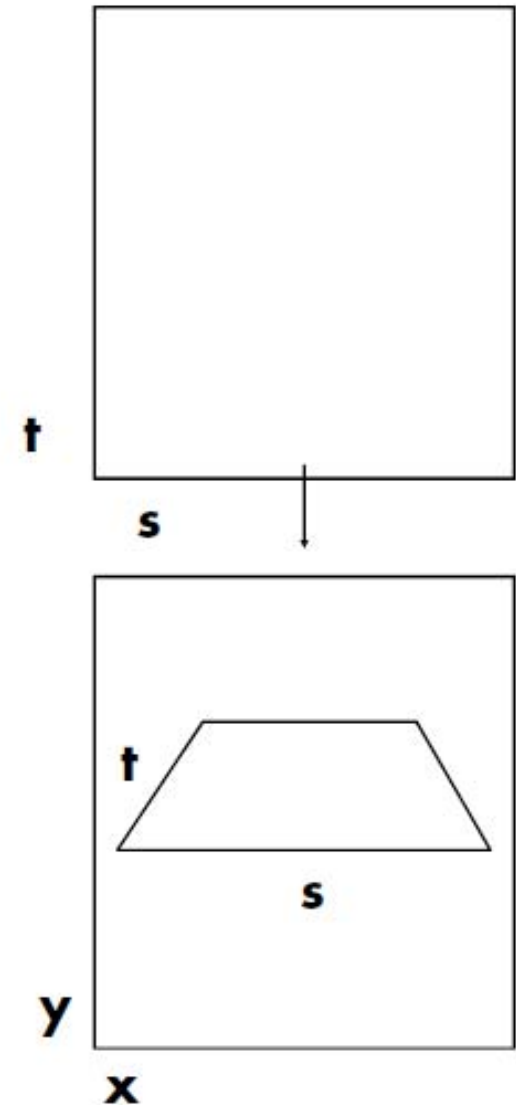
3D World Space

| Viewing Transformation

3D Camera Space

| Projection

2D Image Space





# Texture Mapping Polygons

---

Forward transformation: linear projective map

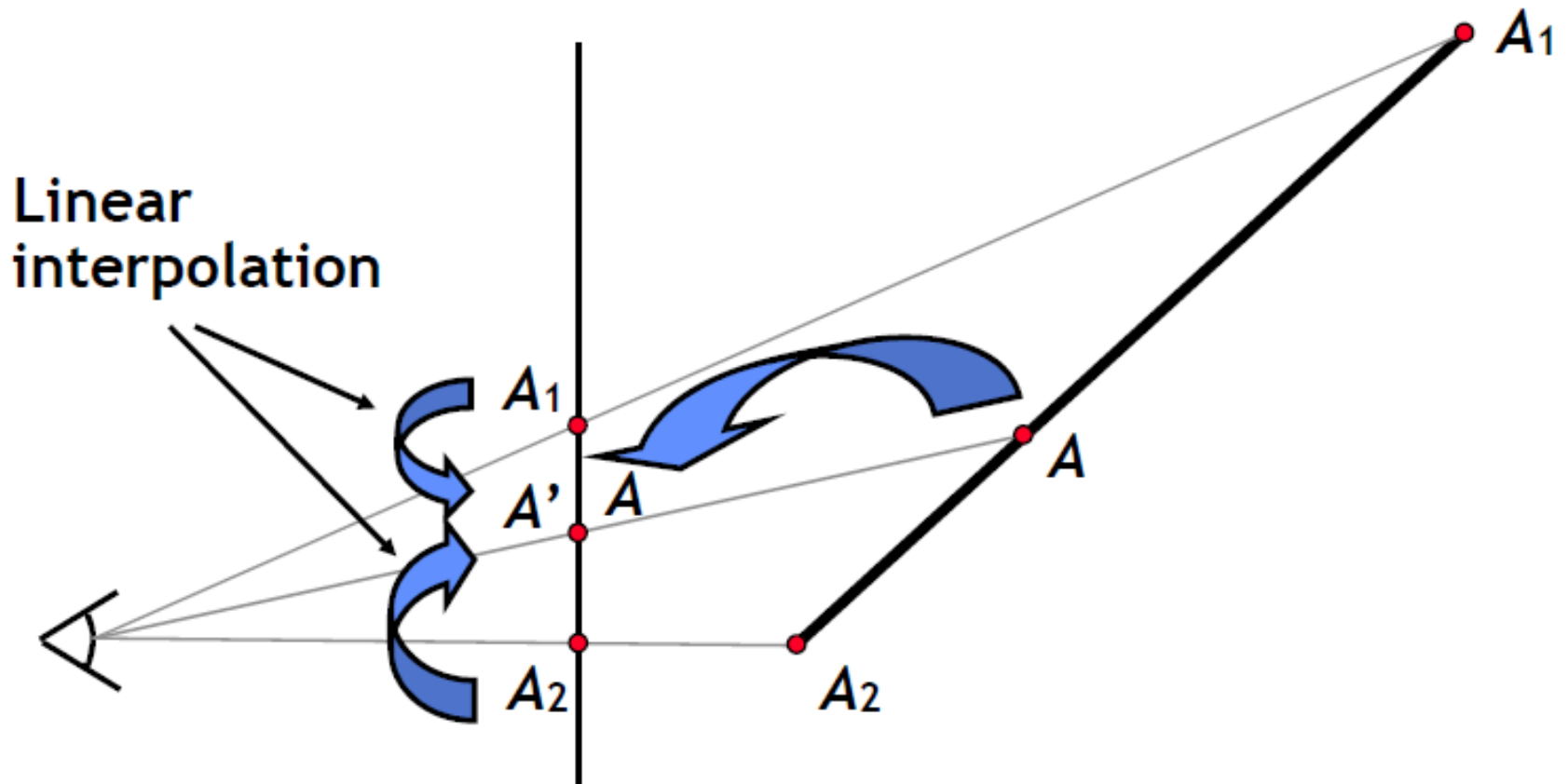
$$\begin{bmatrix} x \\ y \\ w \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \begin{bmatrix} s \\ t \\ r \end{bmatrix}$$

Backward transformation: linear projective map

$$\begin{bmatrix} s \\ t \\ r \end{bmatrix} = \begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix}^{-1} \begin{bmatrix} x \\ y \\ w \end{bmatrix}$$

# Incorrect attribute interpolation

---



$A' \neq A!$

# Linear interpolation

---

Compute intermediate attribute value

- Along a line:  $A = aA_1 + bA_2$ ,  $a+b=1$
- On a plane:  $A = aA_1 + bA_2 + cA_3$ ,  $a+b+c=1$

Only projected values interpolate linearly in screen space (straight lines project to straight lines)

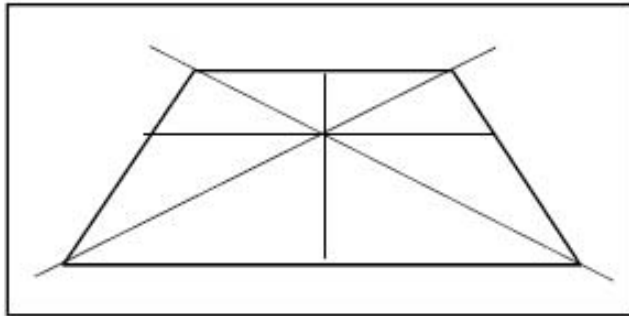
- $x$  and  $y$  are projected (divided by  $w$ )
- Attribute values are not naturally projected

Choice for attribute interpolation in screen space

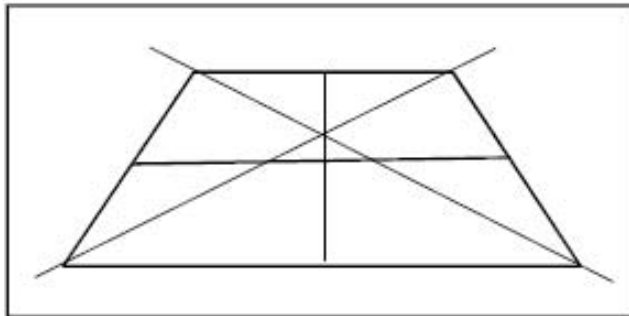
- Interpolate unprojected values
  - Cheap and easy to do, but gives wrong values
  - Sometimes OK for color, but
  - Never acceptable for texture coordinates
- Do it right

# Linear Perspective

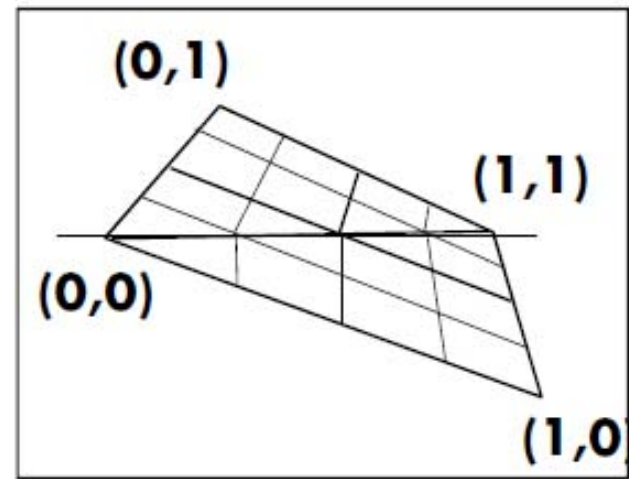
---



**Correct Linear Perspective**



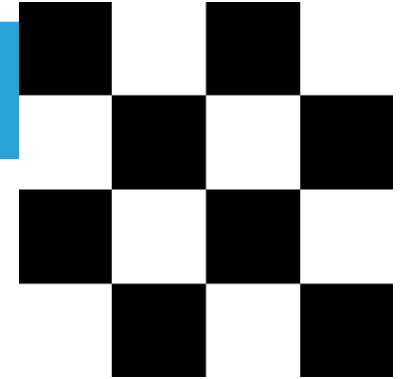
**Incorrect Perspective**



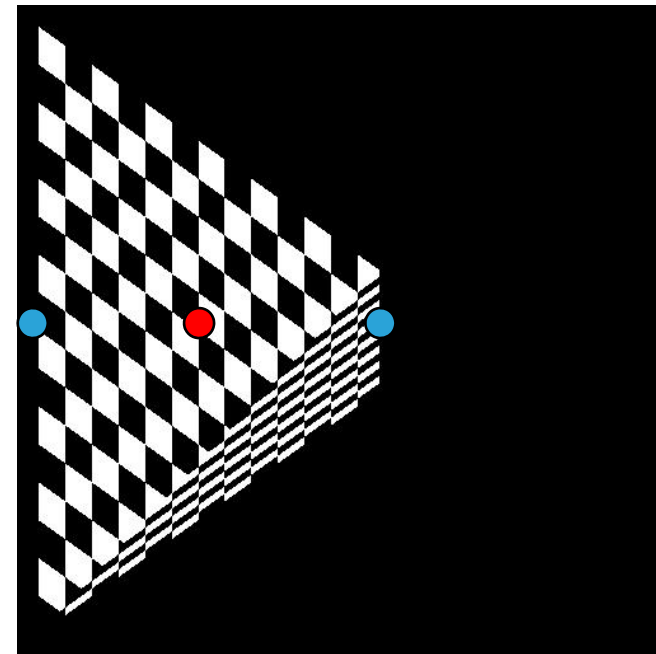
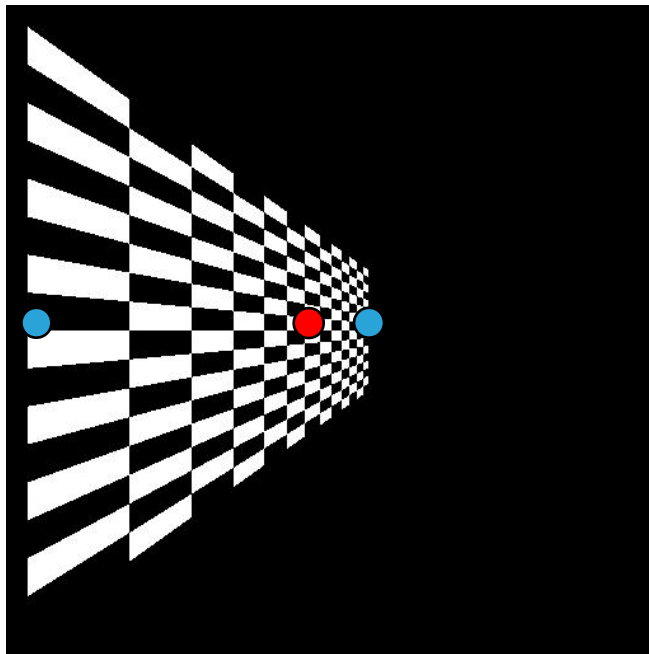
**Linear Interpolation, *Bad***

**Perspective Interpolation, *Good***

# Perspective Texture Mapping



linear interpolation in object space  $\frac{ax_1 + bx_2}{aw_1 + bw_2} \neq a \frac{x_1}{w_1} + b \frac{x_2}{w_2}$  linear interpolation in screen space



$$a = b = 0.5$$



# Early Perspective Texture Mapping in Games



DOOM (id Software, 1993)



# Early Perspective Texture Mapping in Games



Quake (id Software, 1996)

# Perspective-correct linear interpolation

---

Only projected values interpolate correctly, so project  $A$

- Linearly interpolate  $A_1/w_1$  and  $A_2/w_2$

Also interpolate  $1/w_1$  and  $1/w_2$

- These also interpolate linearly in screen space

Divide interpolants at each sample point to recover  $A$

- $(A/w) / (1/w) = A$
- Division is expensive (more than add or multiply), so
  - Recover  $w$  for the sample point (reciprocate), and
  - Multiply each projected attribute by  $w$

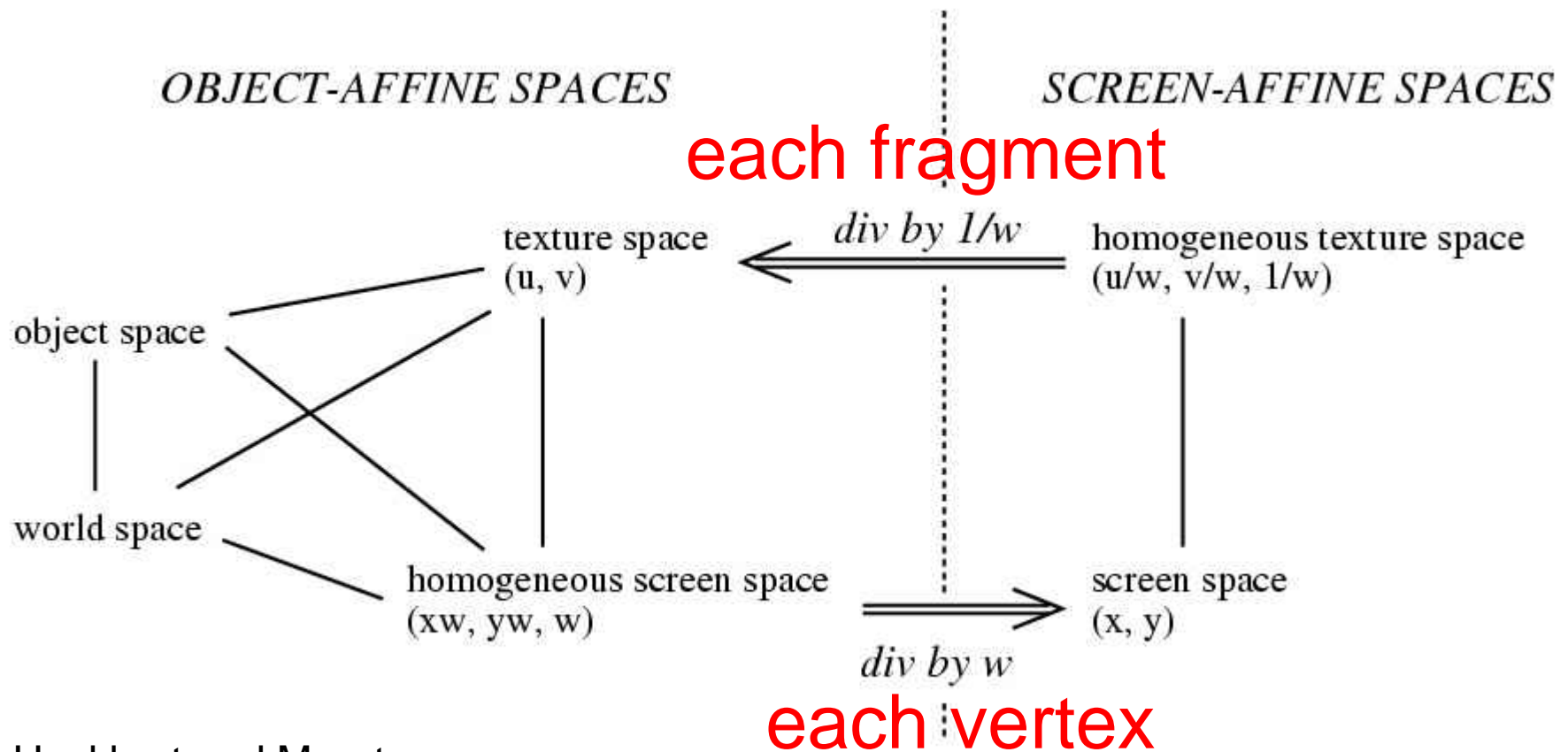
Barycentric triangle parameterization:

$$A = \frac{aA_1/w_1 + bA_2/w_2 + cA_3/w_3}{a/w_1 + b/w_2 + c/w_3} \quad a + b + c = 1$$



# Perspective Texture Mapping

- Solution: interpolate  $(s/w, t/w, 1/w)$
- $(s/w) / (1/w) = s$  etc. at every fragment



Heckbert and Moreton



# Perspective-Correct Interpolation Recipe



$$r_i(x, y) = \frac{r_i(x, y) / w(x, y)}{1 / w(x, y)}$$

- (1) Associate a record containing the  $n$  parameters of interest  $(r_1, r_2, \dots, r_n)$  with each vertex of the polygon.
- (2) For each vertex, transform object space coordinates to homogeneous screen space using  $4 \times 4$  object to screen matrix, yielding the values  $(xw, yw, zw, w)$ .
- (3) Clip the polygon against plane equations for each of the six sides of the viewing frustum, linearly interpolating all the parameters when new vertices are created.
- (4) At each vertex, divide the homogeneous screen coordinates, the parameters  $r_i$ , and the number 1 by  $w$  to construct the variable list  $(x, y, z, s_1, s_2, \dots, s_{n+1})$ , where  $s_i = r_i / w$  for  $i \leq n$ ,  $s_{n+1} = 1 / w$ .
- (5) Scan convert in screen space by linear interpolation of all parameters, at each pixel computing  $r_i = s_i / s_{n+1}$  for each of the  $n$  parameters; use these values for shading.

Thank you.