Computer Graphics Seminar MTAT.03.305

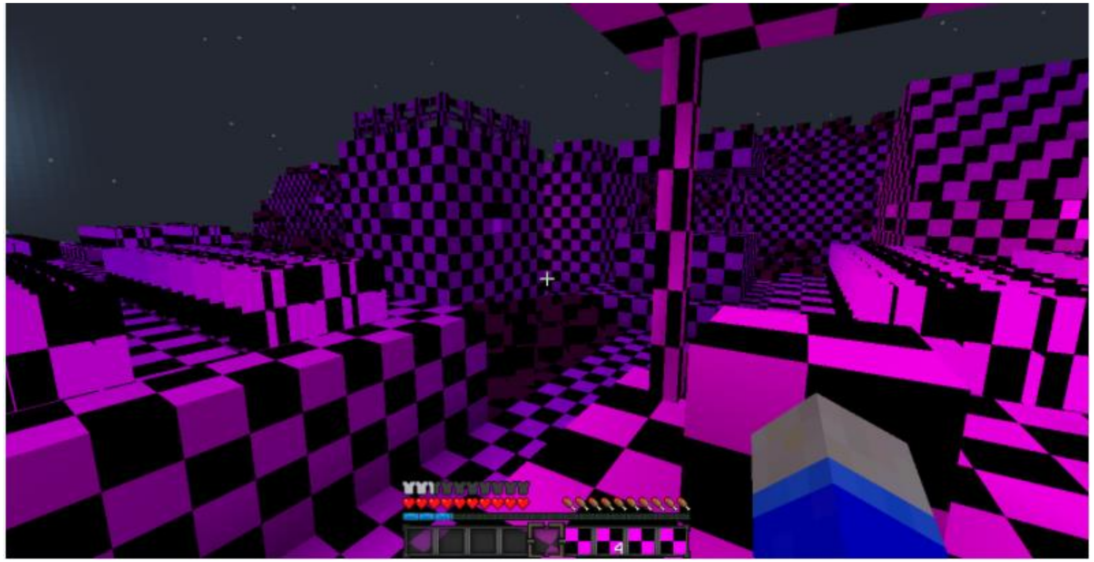
Texture Mapping

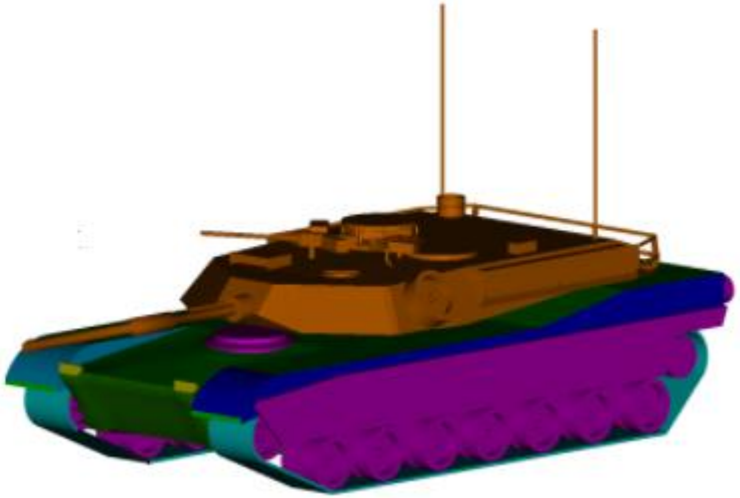
Rauno Näksi

Tartu, 2018

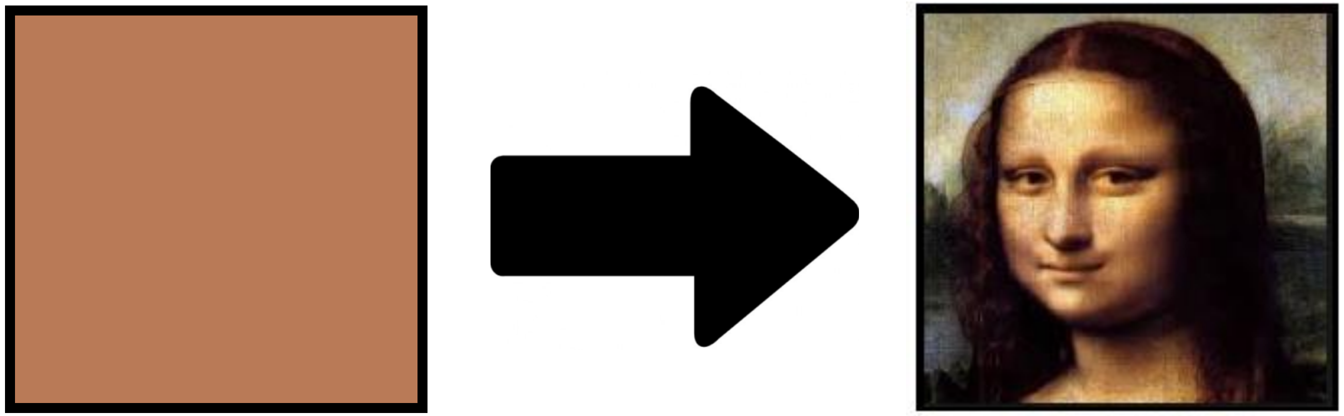
Motivation

- Just colored and shaded surfaces are not realistic



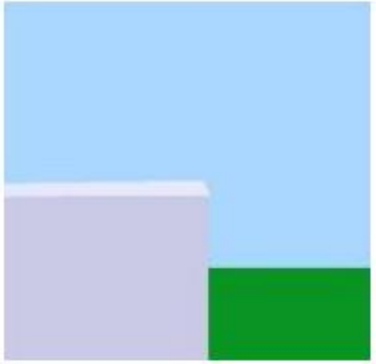


- How to turn



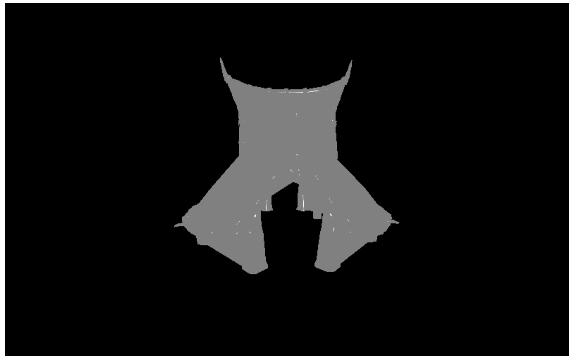
- One option: use a huge number of polygons with appropriate surface coloring and reflectance characteristics
    - Mona Lisa with 50 semi-transparent Polygons

- We can improve the appearance of polygons by mapping images onto their surface



scene without texture

scene with color texture



Goals with Texturing

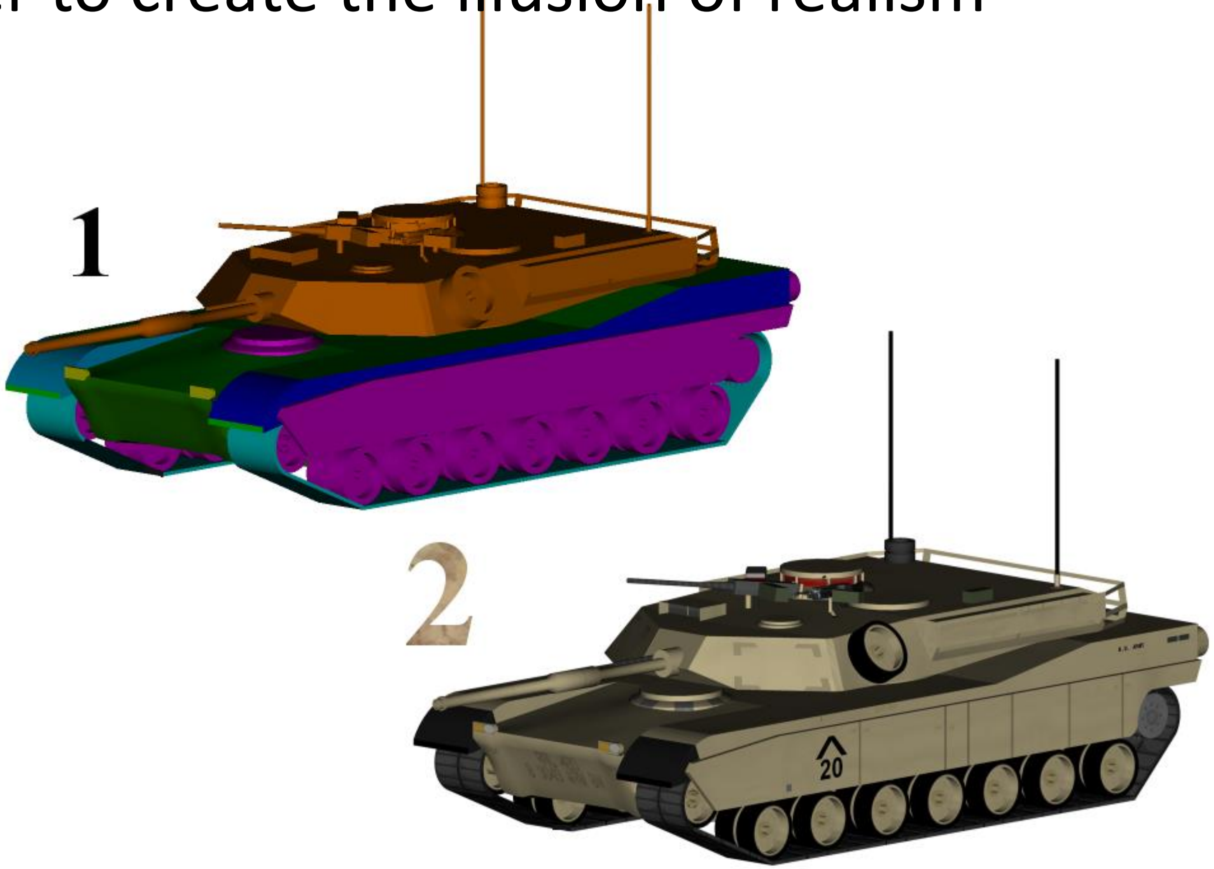
- adding per-pixel surface details without raising the geometric complexity of a scene
- keep the number of vertices and primitives low
- add detail per fragment
- detail refers to any property that influences the final radiance, e.g. object or light properties
- modeling and rendering time is saved by keeping the geometrical complexity low

Types of Mapping

- Texture Mapping
  - Uses images to fill inside of polygons
- Environment (reflection mapping)
  - Uses a picture of the environment for texture maps
  - Allows simulation of highly specular surfaces
- Bump mapping
  - Emulates altering normal vectors during the rendering process
- And more

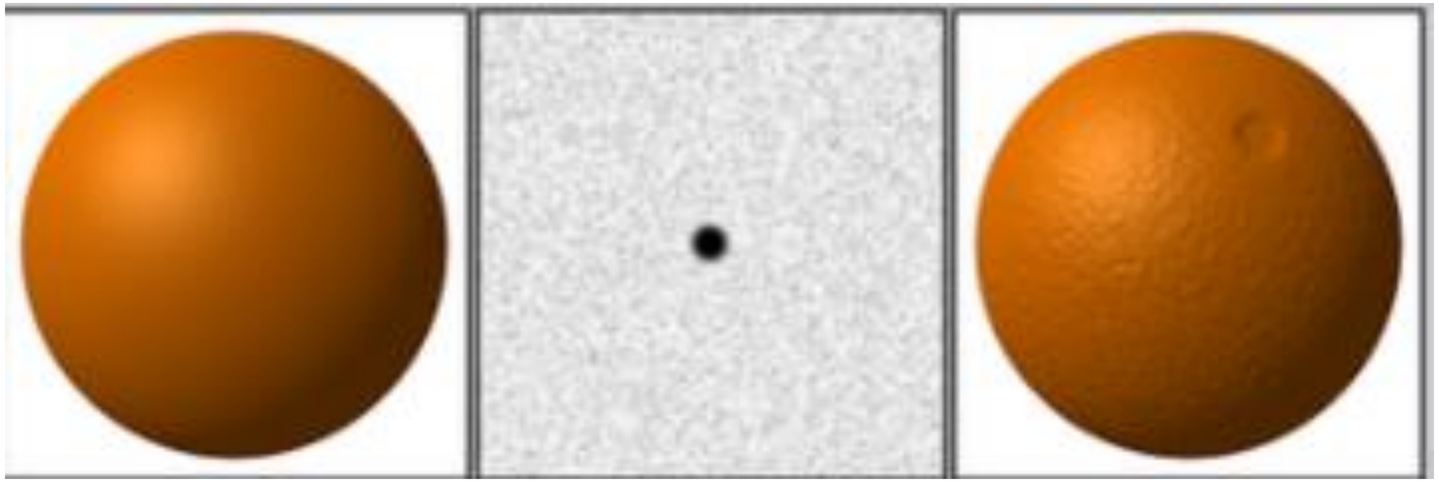
Texture Mapping

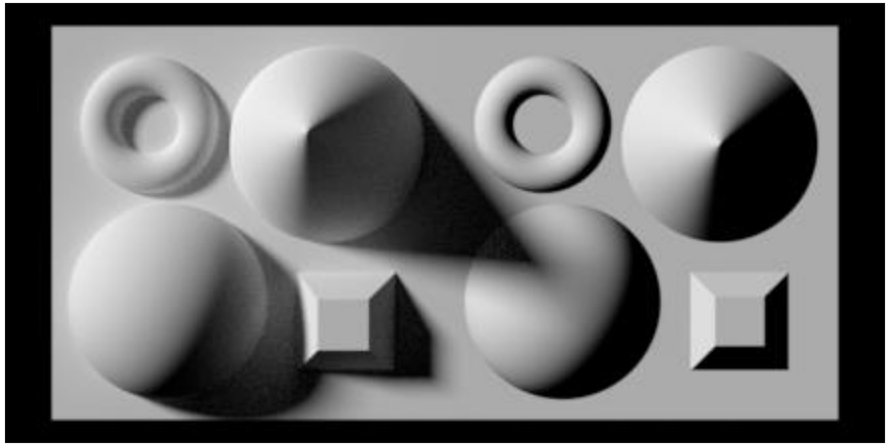
- Instead of calculating color, shade, light, etc. for each pixel we just paste images to our objects in order to create the illusion of realism



Environment (reflection mapping)

Bump mapping





Texture Mapping

Texture mapping

- Adding lots of detail to our models to realistically depict skin, grass, bark, stone, etc., would increase rendering times dramatically, even for hardware-supported projective methods.

Texture mapping

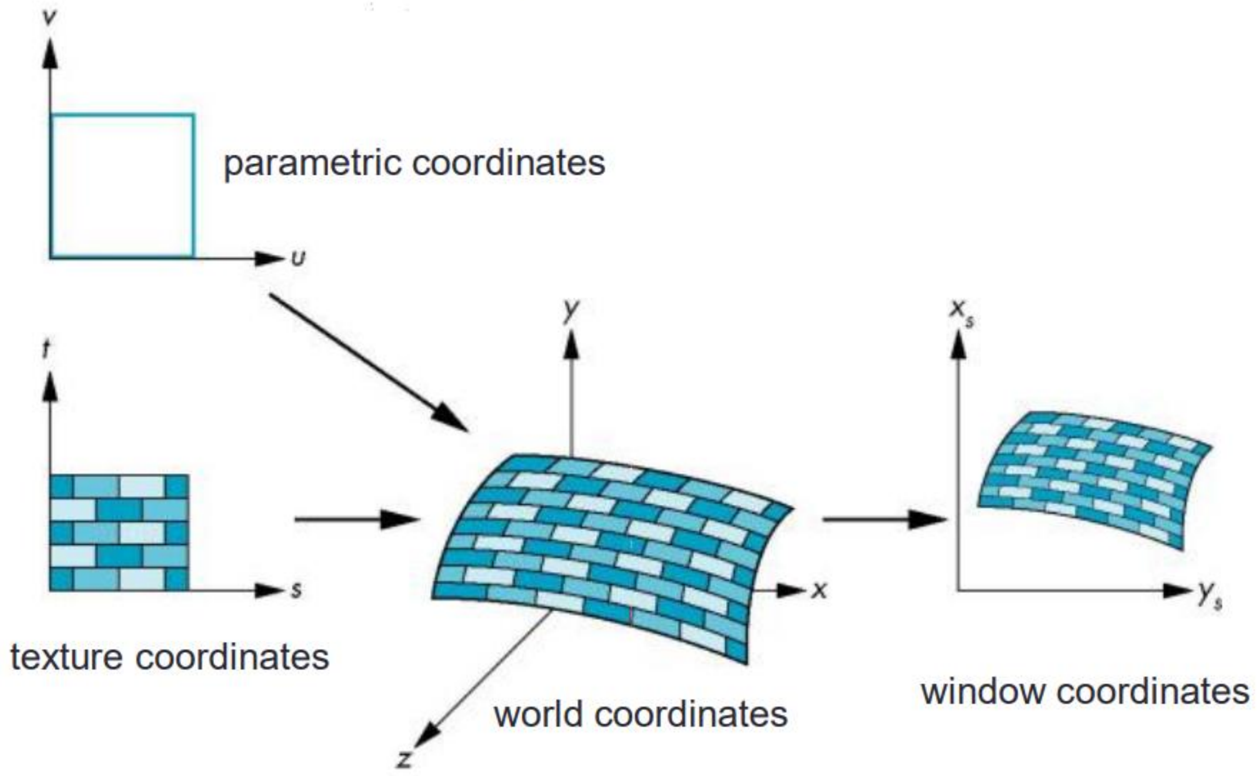
- Adding lots of detail to our models to realistically depict skin, grass, bark, stone, etc., would increase rendering times dramatically, even for hardware-supported projective methods.

Concept

- 2D textures are represented as 2D images
- textures can store a variety of properties, i.e. colors, normals
- positions of texture pixels, i. e. texels, are characterized by texture coordinates (u, v) in texture space
- texture mapping is a transformation from object space to texture space (x, y, z) -> (u, v)
  - texture coordinates (u, v) are assigned to a vertex (x, y, z)
- texture mapping is generally applied per fragment
  - rasterization determines fragment positions and interpolates texture coordinates from adjacent vertices
  - texture lookup is performed per fragment using interpolated texture coordinates

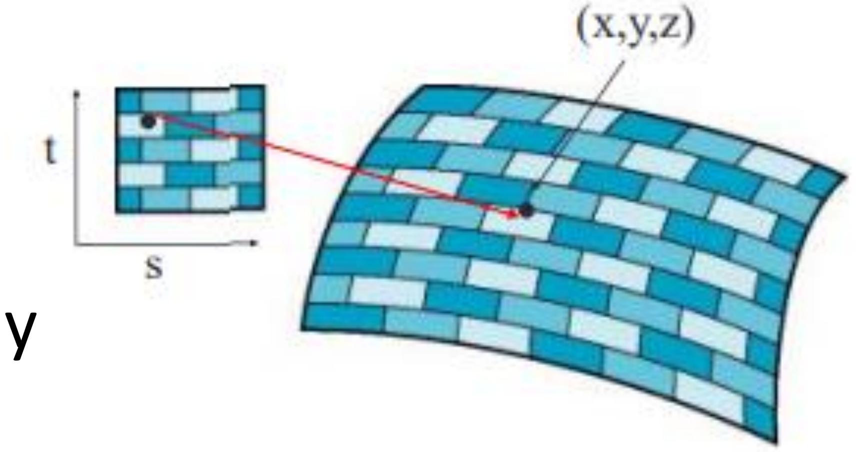
Coordinate Systems

- Parametric coordinates
  - May be used to model curves and surfaces
- Texture coordinates
  - Used to identify points in the image to be mapped
- Object or World Coordinates
  - Conceptually, where the mapping takes place
- Window Coordinates
  - Where the final image is really produced


v
u
parametric coordinates
t
s
texture coordinates
y
x
z
world coordinates
x_s
y_s
window coordinates

Mapping Functions

- Basic problem is how to find the maps
- Consider mapping from texture coordinates to a point a surface
- Appear to need three functions
  - x = x(s,t)
  - y = y(s,t)
  - z = z(s,t)
- But we really want to go the other way

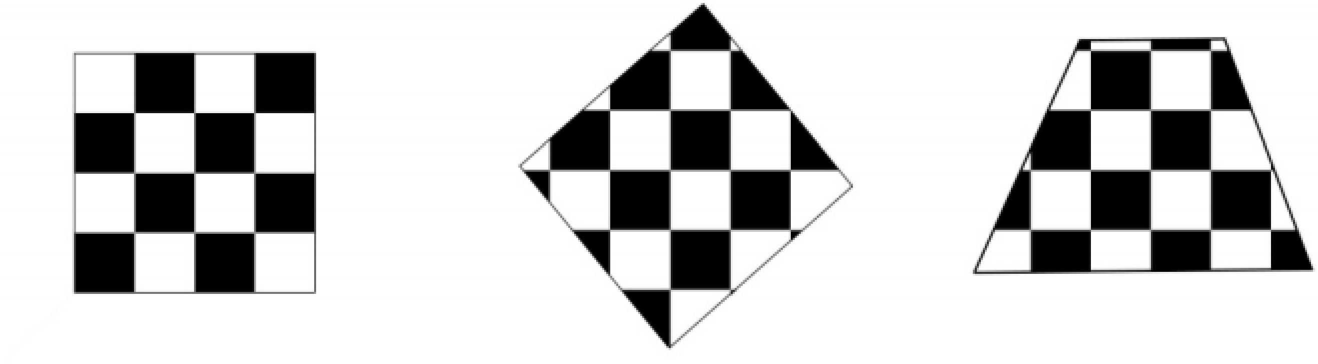


Backward Mapping

- We really want to go backwards
  - Given a pixel, we want to know to which point on an object it corresponds
  - Given a point on an object, we want to know to which point in the texture it corresponds
- Need a map of the form
  - $s = s(x,y,z)$
  - $t = t(x,y,z)$

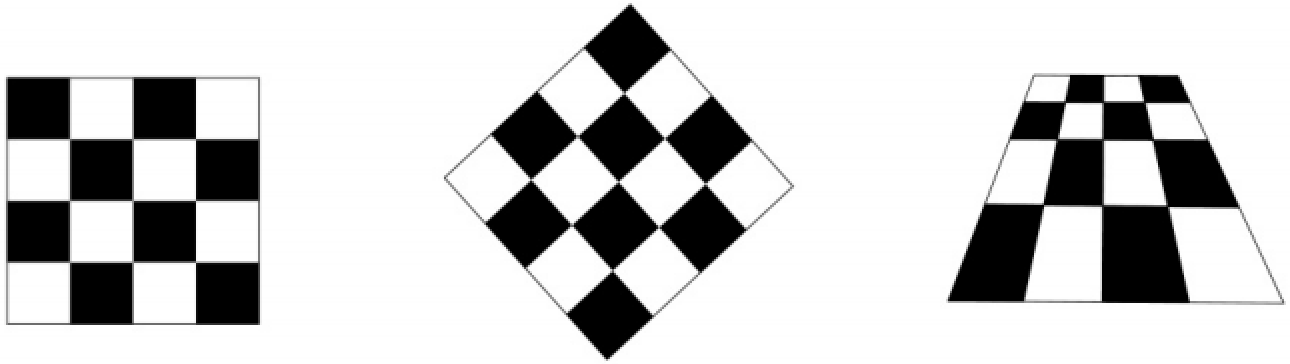
Non-parametric texture mapping

- With “non-parametric texture mapping”
  - Texture size and orientation are fixed
  - They are unrelated to size and orientation of polygon



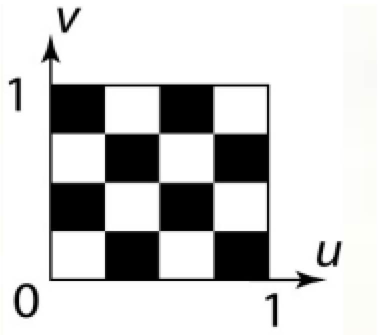
Parametric texture mapping

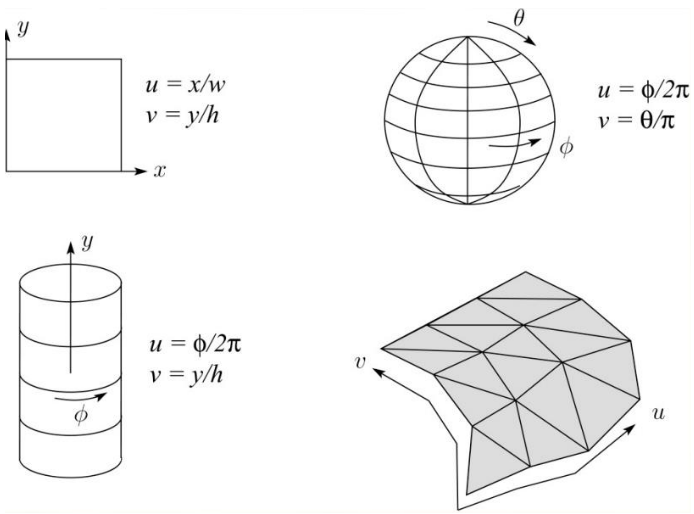
- With “parametric texture mapping,” texture size and orientation are tied to the polygon.
- Idea
  - Separate “texture space” and “screen space”
  - Texture the polygon as before, but in texture space
  - Deform (render) the textured polygon into screen space
- A texture can modulate just about any parameter – diffuse color, specular color, specular exponent, …



Implementing texture mapping

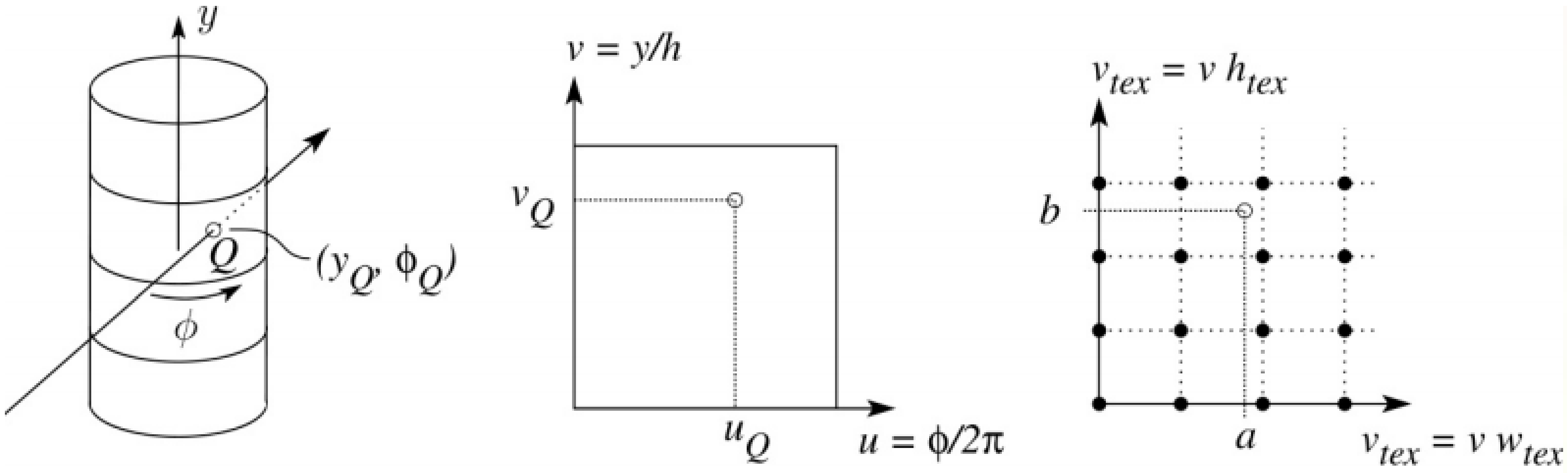
- To apply textures we need 2D coordinates on surfaces
  - Parameterization
- A texture lives in it own abstract image coordinates parameterized by (u,v) in the range ([0..1], [0..1])
- Some objects have a natural parameterization





Mapping to texture image coordinates

- The texture is usually stored as an image. Thus, we need to convert
  - from abstract texture coordinate (u,v) in the range ([0..1], [0..1])
  - to texture image coordinates (u_{tex}, v_{tex}) in the range ([0..w_{tex}], [0.. h_{tex}])

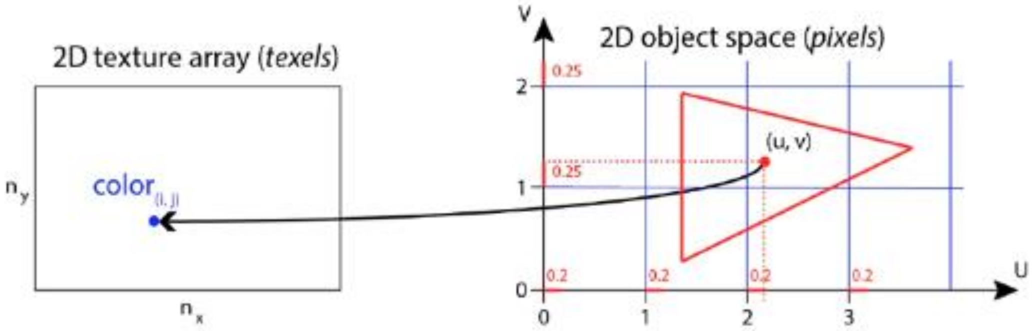


Ray intersection

Mapping to abstract texture coords

Mapping to texture pixel coords

- What happens if object space (pixels) is between texture array (texels)?



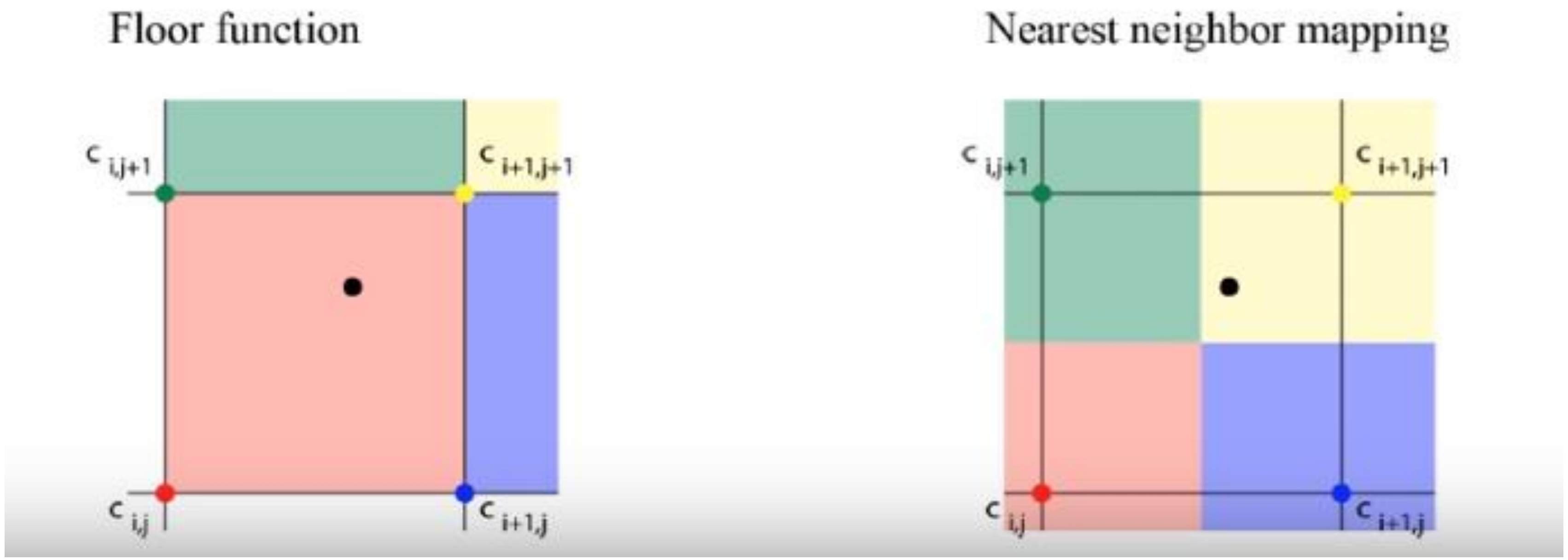
The pixel (i, j) in the $n_x \times n_y$ image for (u, v) is found by

$$i = \lfloor un_x \rfloor \ and \ i = \lfloor vn_y \rfloor$$

$\lfloor x \rfloor$ is the floor function that give the highest integer value ≤ x.

Nearest neighbor interpolation

- This is a version of nearest-neighbor interpolation, because we take the color of the nearest neighbor

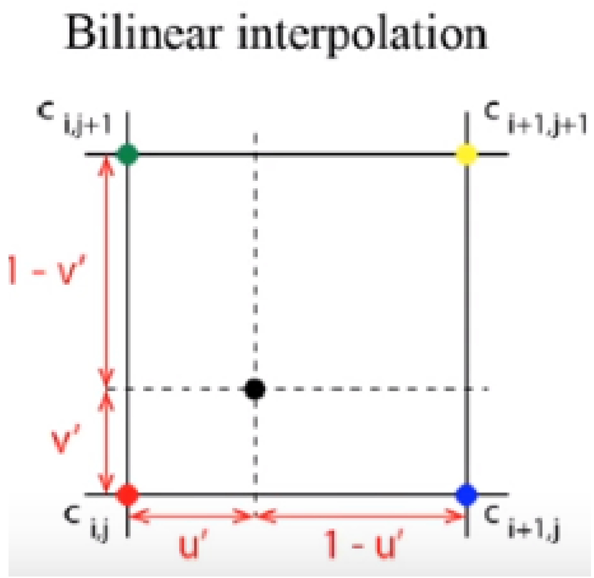


- For smoother effects we may use bilinear interpolation

$$c(u,v) =$$

$$(1-u')(1-v')c_{ij} + u'(1-v')c_{(i+1)j} + (1-u')v'c_{i(j+1)} + u'v'c_{(i+1)(j+1)},$$

where $u' = un_x - \lfloor un_x \rfloor$ and $v' = vn_y - \lfloor vn_y \rfloor$



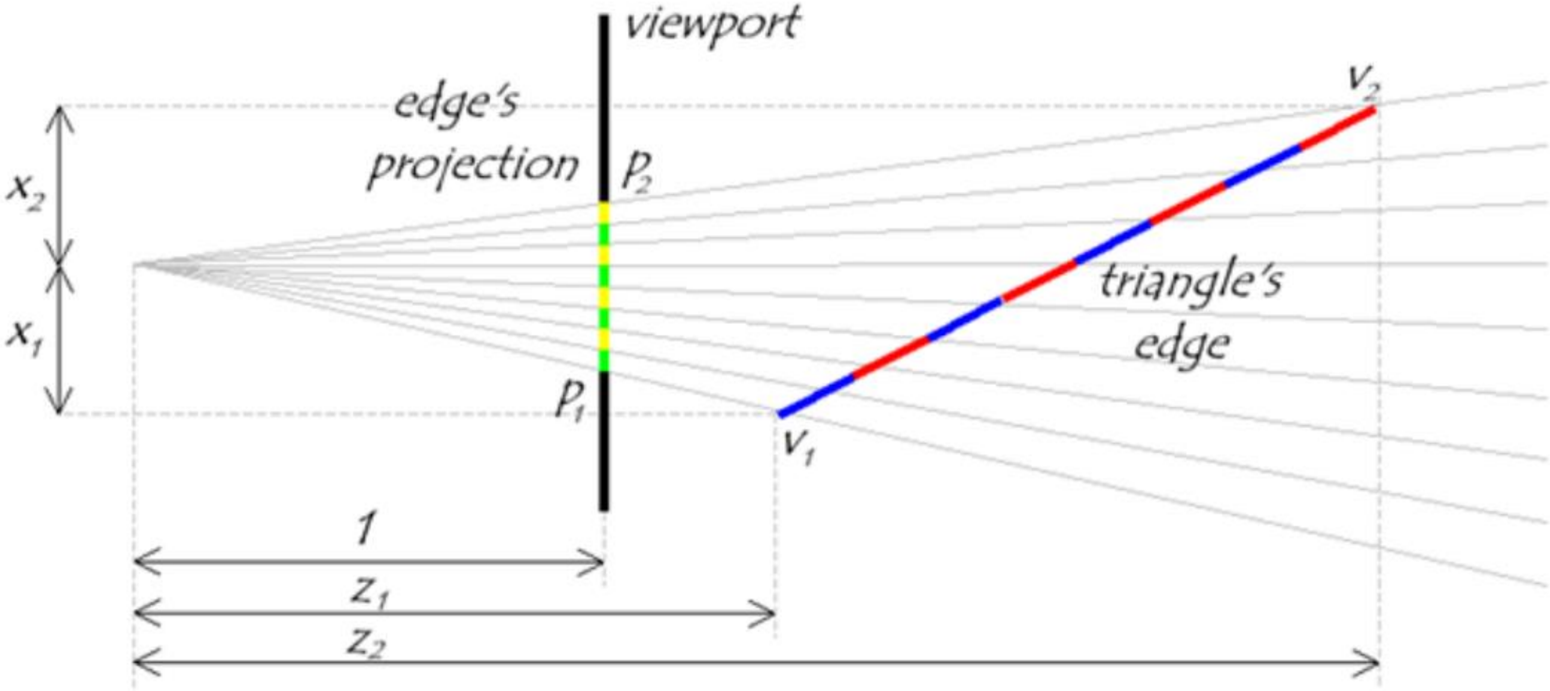
Linear interpolation

- Linearly interpolating uv coordinates does not produce the expected results

texture source | what we get | what we want

Why does this happen?

- Uniform steps in 2D screen space do not correspond to uniform steps over the surface of the triangle



Texturing 3D objects

- 3D mapping, which is a procedural approach, i.e. we use a mathematical procedure to create a 3D texture, i.e.
$$f(x, y, z) = c \; with \; c \in \mathbb{R}^3$$
- Then we use the coordinates of each point in our 3D model to calculate the appropriate color value using that procedure, i.e.

$$f(x_p, y_p, z_p) = c_p$$

3D stripe textures

- A simple example: stripes along the X-axis

$stripe\ (x_p, y_p, z_p)\{$

$if(sinx_p > 0)$

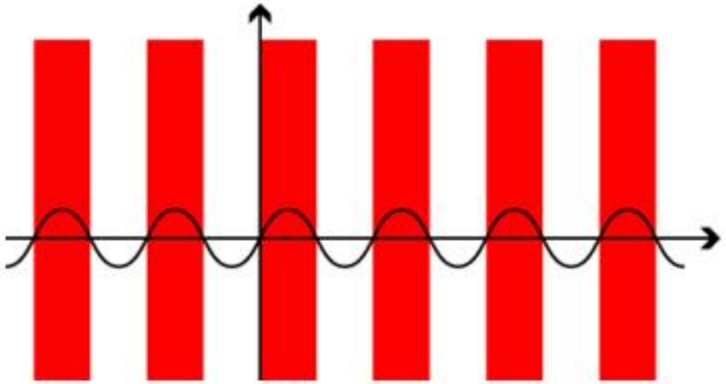
return color0;

else

return1;

}

}



3D stripe textures

- A simple example: stripes along the X-axis

$stripe\left(x_p, y_p, z_p\right)\{$

$if\left(sin x_p > 0\right)$

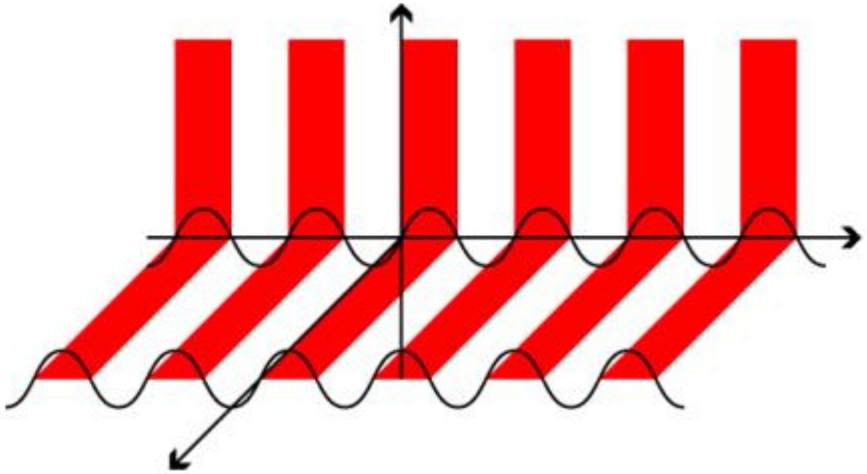
return color0;

else

return1;

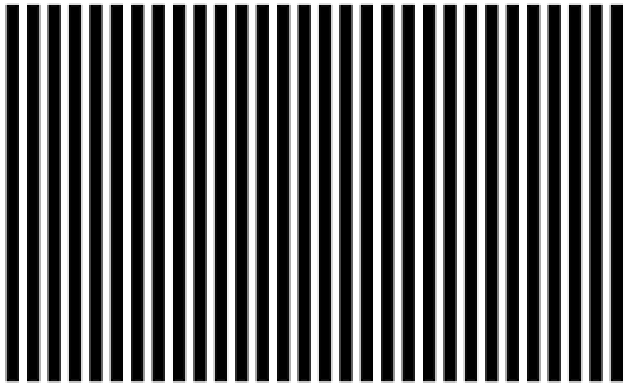
}

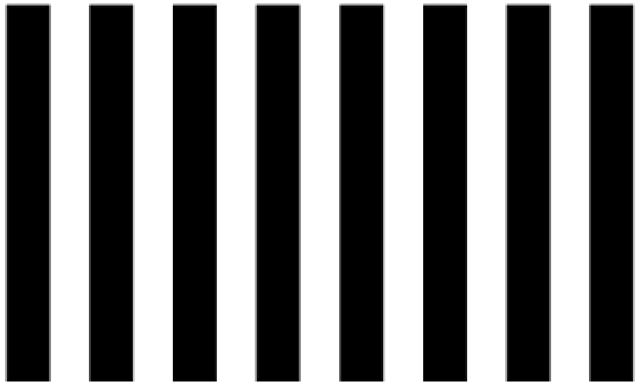
}



- Stripes with controllable width

```
stripe (point p, real width){
        if(sin(πx_p/width) > 0)
                return color0;
        else
                return1;
        }
}
```





- Stripes with controllable width

$stripe\ (point\ p, real\ width)\{$

$if\left(sin(\pi x_p/width) > 0\right)$

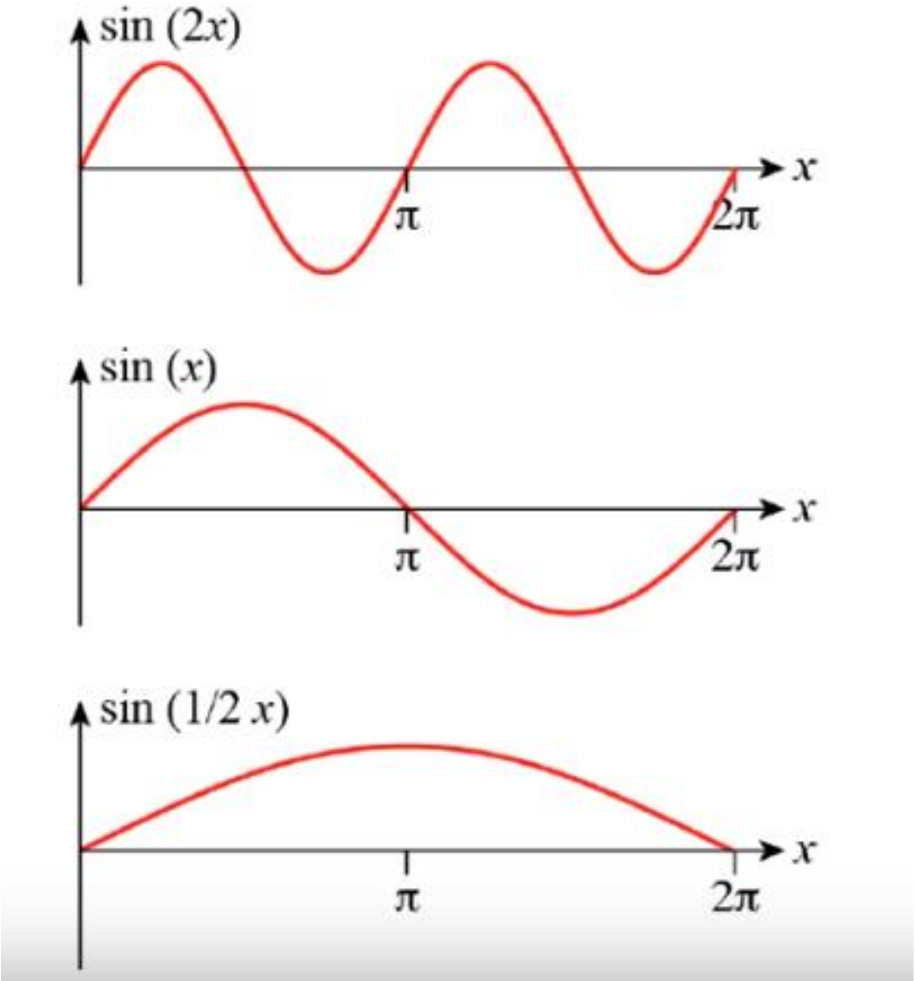
return color0;

else

return1;

}

}



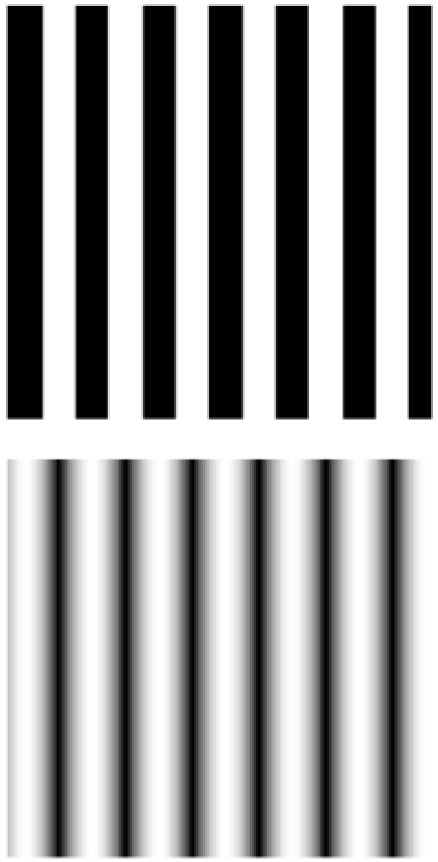
- Smooth variation between two colors, instead of two distinct ones

$stripe\ (point\ p, real\ width)\{$

$$t = (1 + \sin\left(\frac{\pi x_p}{width}\right))/2$$

return $(1 - t)c_0 + tc_1$

$\}$



References

- http://www.inf.ed.ac.uk/teaching/courses/cg/lectures/slides8.pdf
- https://en.wikipedia.org/wiki/Texture_mapping
- http://courses.cs.vt.edu/~cs4204/lectures/texture_mapping.pdf
- https://www.cs.utexas.edu/users/fussell/courses/cs384g-fall2011/lectures/lecture12-Texture_mapping.pdf
- http://www.cs.uu.nl/docs/vakken/gr/2011/Slides/06-texturing.pdf
- http://www.sci.tamucc.edu/~sking/Courses/COSC4328/Notes/Textures-6.pdf
-