Basic Techniques in Computer Graphics

Winter 2017 / 2018

The slide comments are not guaranteed to be complete, they are no alternative to the lectures itself. So go to the lectures and write down your own comments!

Overview

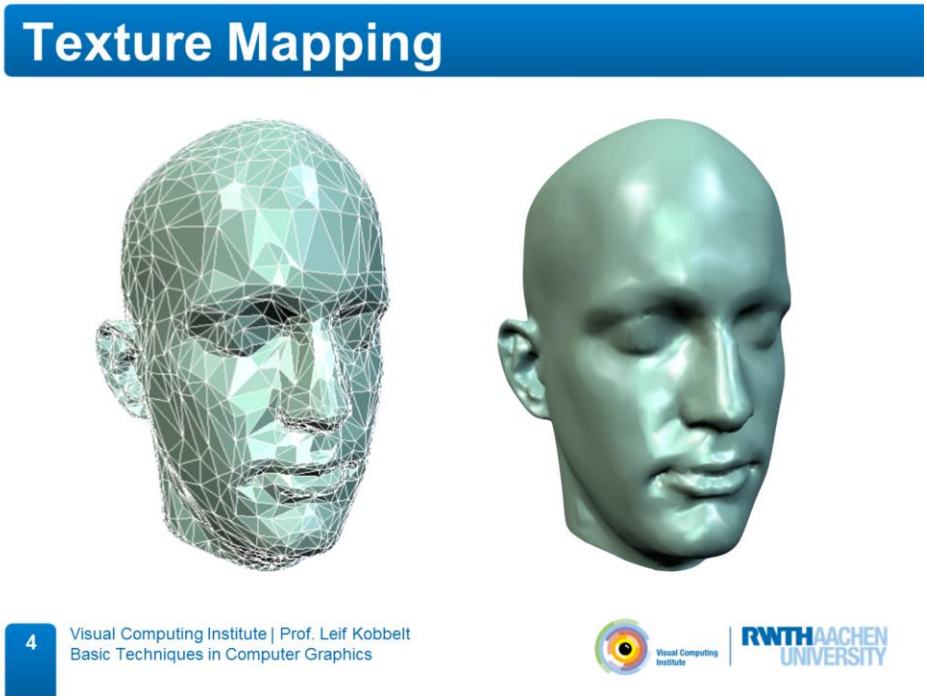
- **Texture Mapping**
- Texture Anti-Aliasing
- Geometry Anti-Aliasing

Texture mapping can introduce high frequency changes of material properties without adding more geometry.

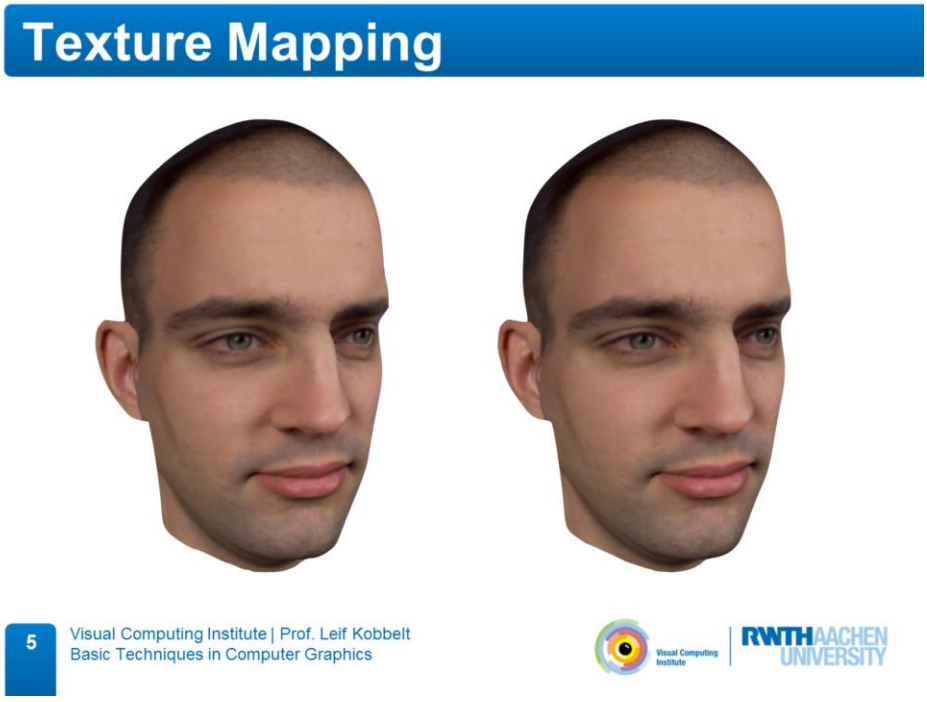
Texture Mapping

- Paste images on surfaces
- Assign individual material properties to every point / pixel on a surface
- Assignment by global surface parameterization

Textures are like wallpapers which can be placed on the geometry to simulate more surface details. Traditionally textures were used for the diffuse color of an object but when more complex materials are simulated they can hold arbitrary material properties.



A low resolution mesh (left) and a high resolution mesh (right). Note that the mesh on the right side is also shaded with Gouraud Shading instead of flat shading on the left (one color per triangle).



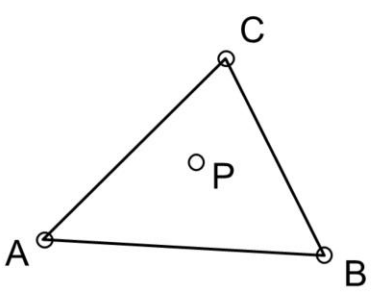
Adding a texture to the meshes from the previous slide adds more details. Now both meshes look like they have the same amount of detail.

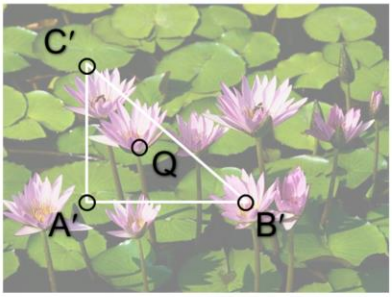
Texturing one Triangle

- Texture coordinates for every vertex
- Linear interpolation
  - Image space
  - Object space
- Color look-up in texture

Every vertex gets one texture coordinate (in most cases a 2D coordinate). While rasterizing the triangle, the texture coordinate of the three vertices get interpolated over the surface. Each generated fragment can look up a (color) value from the texture based on its individual texture coordinate.

Texturing one Triangle



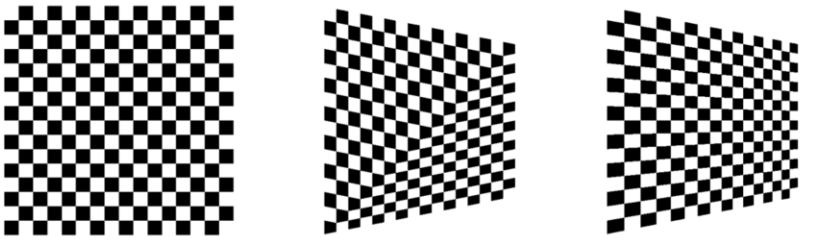


$P = \alpha A + \beta B + \gamma C$
$\in R^{2 \text{ or } 3}$

$Q = \alpha A' + \beta B' + \gamma C'$
$\in R^2$

For every vertex additional coordinates in texture space can be defined. A point inside of the triangle is defined by the barycentric coordinates (left). The same coordinates can be used to interpolate the triangles texture coordinate (or any other attribute).

Texturing one Triangle



Perpendicular | Screen space interpolation | Object space interpolation

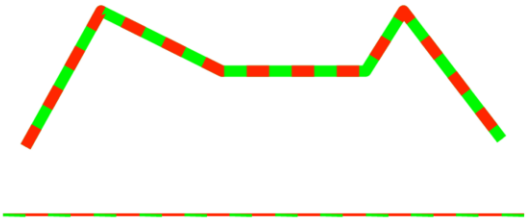
Middle: Screen space interpolation of texture coordinates will introduce deformations (parallel lines stay parallel).

Texture Distortion

Top: mesh; Bottom: texture.

A naive parallel projection can introduce distortion which should be minimized.

Texture Distortion



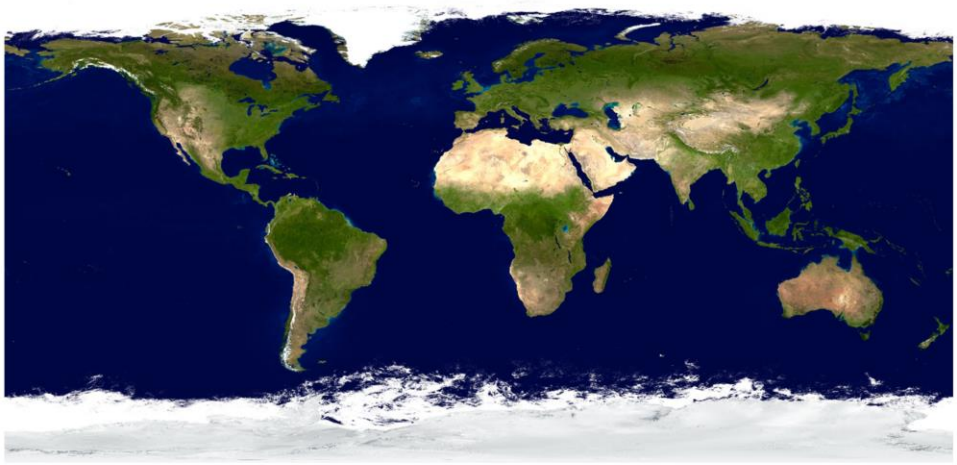
Better way: first map the images to the geometry and than flatten it down. This can become a quite complex task for general meshes.

Texturing an Entire Object

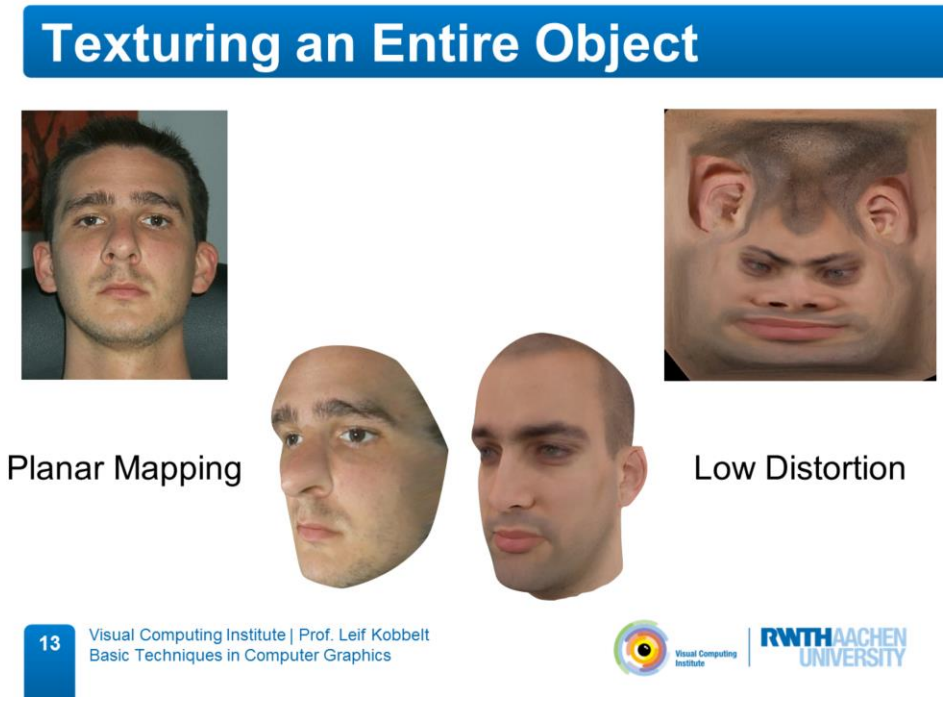
- Consistent texture coordinates for the vertices (shared interior vertices)
- Global parameterization
  - Cylinder mapping
  - Sphere mapping
  - Cube mapping
  - Projective Mapping
  - Low-distortion mapping (spring-models)

Simple objects can be projected onto a similar proxy geometry. Not all of these can be mapped without distortions.

Spherical Parameterization



A mapping onto a sphere can‘t be done without distortion.

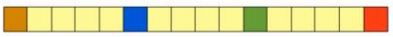


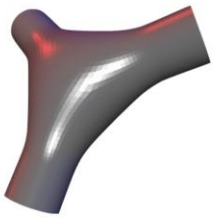
Left: A planar mapping of a photo onto a mesh gets distorted towards the edges of the mesh.

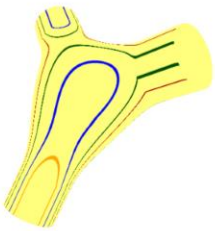
Right: Low distortion on the mesh can result to high distortion on the texture (which makes it hard for artists to create/change it). Often a texture has to be cut open into multiple parts as well.

Special Texture Maps

- 1D Textures:
  - Non-linear interpolation
  - Isolines (e.g. isophotes)



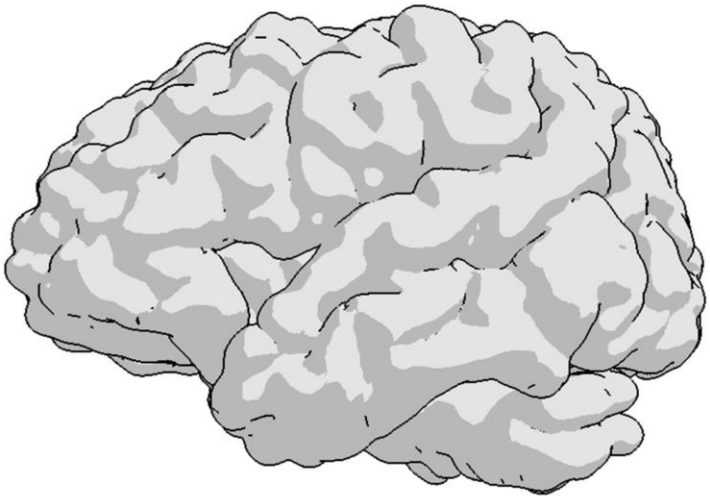




- 3D Textures
  - Model volumetric materials (e.g. wood)

The angle between the normal and the light source can get interpreted as a texture coordinate into a 1D texture to visualize these isophotes (german: Isolinie) or contour line.

Special Texture Maps



1D texture lookup for toon-shading

Color Mapping

- pre-classification
  - determine color per vertex
  - interpolate across polygons
  - high efficiency
- post-classification
  - interpolate texture coordinates
  - texture lookup per fragment
  - „sharper“ contours

The visual quality of pre-classification will be the same as having vertex attributes instead of a texture. Today, post-classification is used most of the time to get an optimal quality.

Special Texture Maps

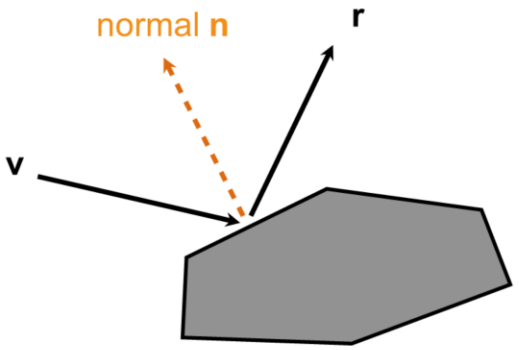
Use texturing for pseudo reflection:

- Light mapping
  - precompute complex lighting
- Reflection mapping
  - view environment from within the object
  - use image as textures (e.g. cube)
- Environment mapping
  - precompute reflected environment
  - use reflected vector to access texture

Environment Mapping

Correct way of calculating reflections:

- Calculate reflection vector r
- Ray-Cast in the scene



v is the normalized vector from the eye to the object
n the normalized surface normal
r the normalized reflection vector of v

Environment Mapping

Observation:

- The Ray-Cast will return **similar** colors for the **same** reflection vector if the object is small (compared to the scene size).

→ Pre-calculate all possible reflections from one position!

If the object is small and the scene is large (like a teapot in a room), the reflection is more dependent of the reflection vector than of the position on the object. Also, if the object moves in a large environment the changes in the reflection are small.
If the object is not perfect reflecting (like a mirror), minor errors in the reflection are harder to notice. So to speed up the rendering a mapping of the whole environment (without the object itself) from one point of view in a special texture can be used.

Environment Mapping

Environment Maps:

- Blinn & Newell's Method
- Sphere Mapping
- Cube Mapping
- Parabolic Mapping

Terminator 2 (1991)

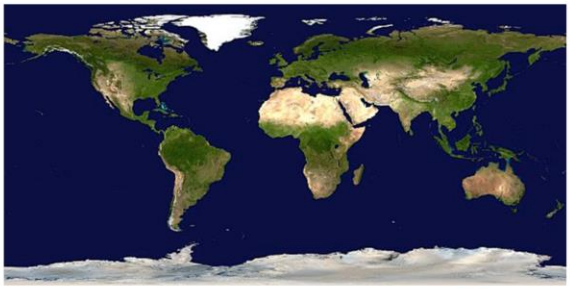
The different kinds of maps differ in the way they can be created and how look ups can be performed.

Blinn & Newell

Idea:

- Map the surrounding world to one image
- Each view direction is defined by two angles

→ Similar to the unwrapping of the earth on a map!



The first environment mapping algorithm was developed by Blinn and Newell in 1976. To map all possible angles from a point into the scene, a similar mapping was proposed as used for unwrapping a map of the globe.
Construction and usage of this kind of map has its drawbacks so this kind of environment mapping is not very popular.

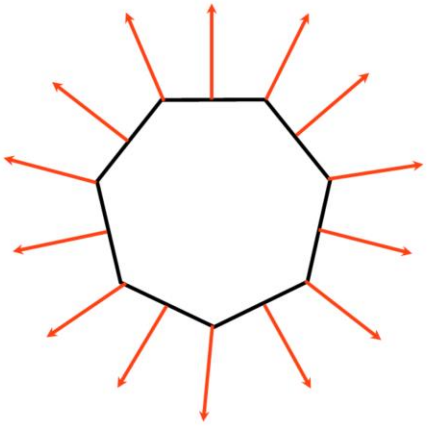
Sphere Mapping

Idea:

- Map the surrounding world to one photo
- Photograph a reflecting sphere

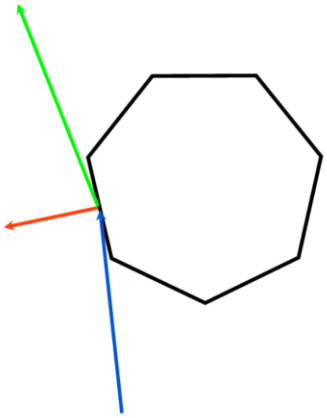
On the pro side with this idea it‘s possible to create environment maps from real scenes without special equipment (a digital camera and a reflecting ball from a christmas tree will do).
On the other hand this map is view dependent as it maps the front facing part of the scene well but get disturbed at the edges (and the photographer is always part of the map).
Only a very small part of the environment hidden behind the sphere is missing in the map.

Sphere Mapping

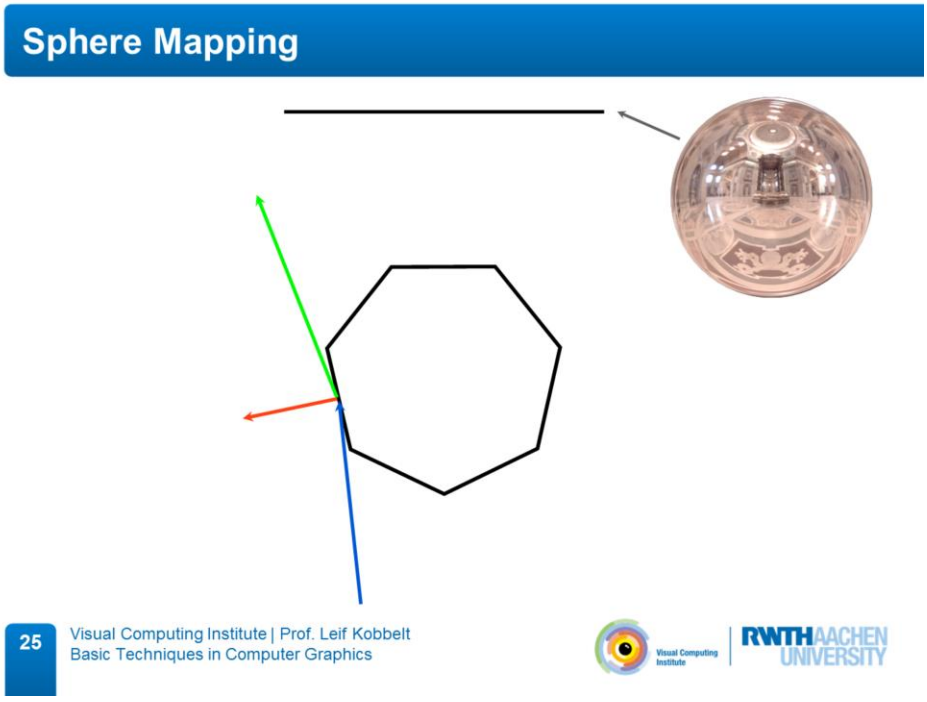


A sphere build from a few polygons is shown with vertex normals and some interpolated normals along the surfaces (red).

Sphere Mapping

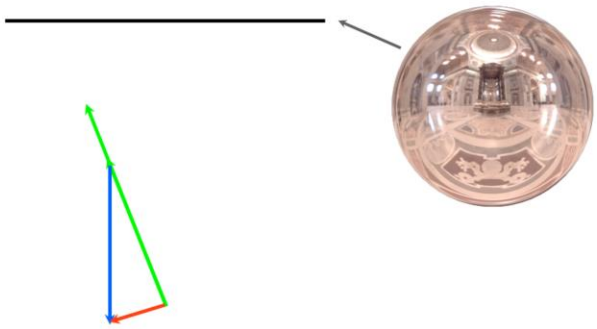


The vector from the camera to a surface point is shown in blue. The ray gets reflected at the surfaces (based on the normal), the reflected vector is shown in green.



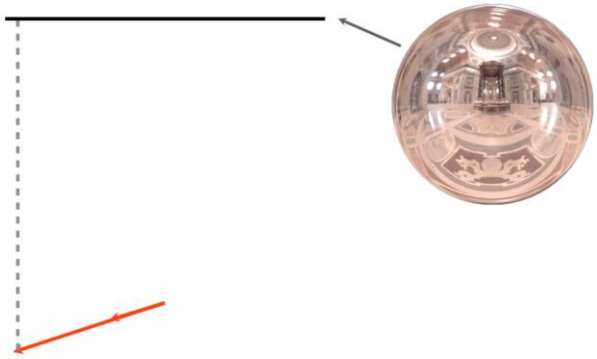
The green reflected vector needs to get mapped to the sphere map shown above.

Sphere Mapping



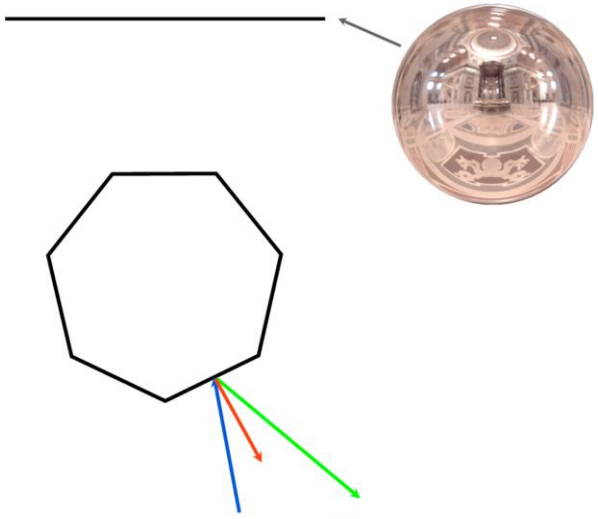
First: normalize the reflected vector.
Add (0,0,1) (blue vector). The result is the red vector.

Sphere Mapping



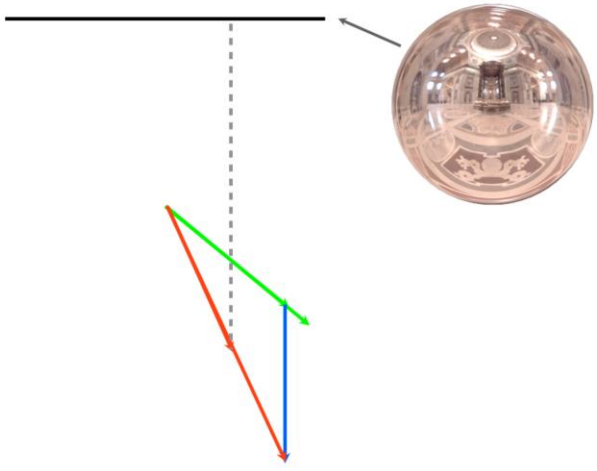
Normalize the red vector.
The x,y component will define the sphere map texture coordinate. Note that texture coordinates range from 0..1 so a mapping from -1..1 to 0..1 has to be performed before the lookup!

Sphere Mapping



A second example.

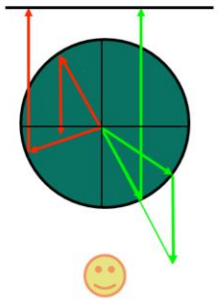
Sphere Mapping



A second example.

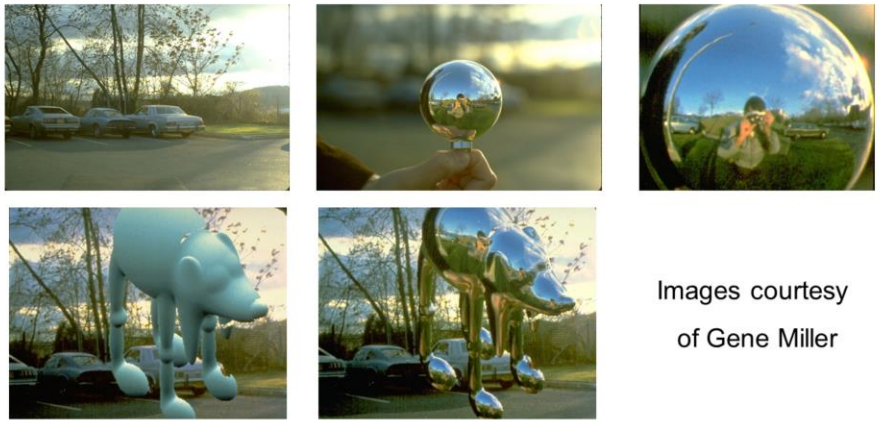
EM: Sphere Mapping

- Normalized reflected vector: [x, y, z]
- Constant shift [x, y, z+1]
- Renormalize [x´, y´, z´]
- Project [x´, y´]



The smilie represents the camera.

EM: Sphere Mapping



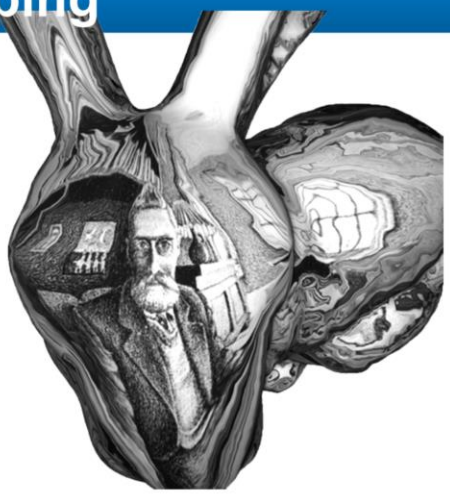
Images courtesy of Gene Miller

Example of sphere maps.

EM: Sphere Mapping

Another example.

EM: Sphere Mapping

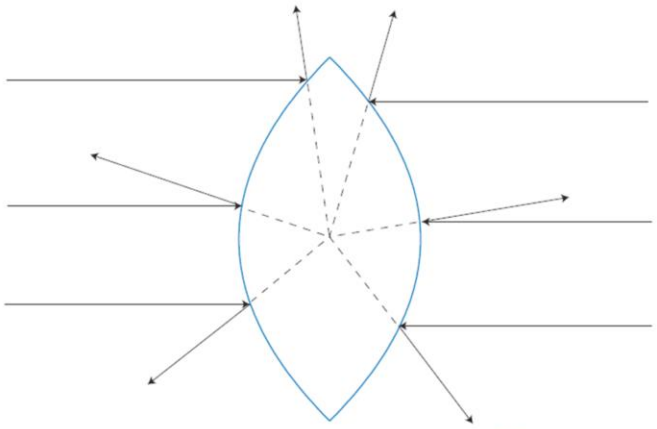


Escher: Hand with reflecting sphere (1935)

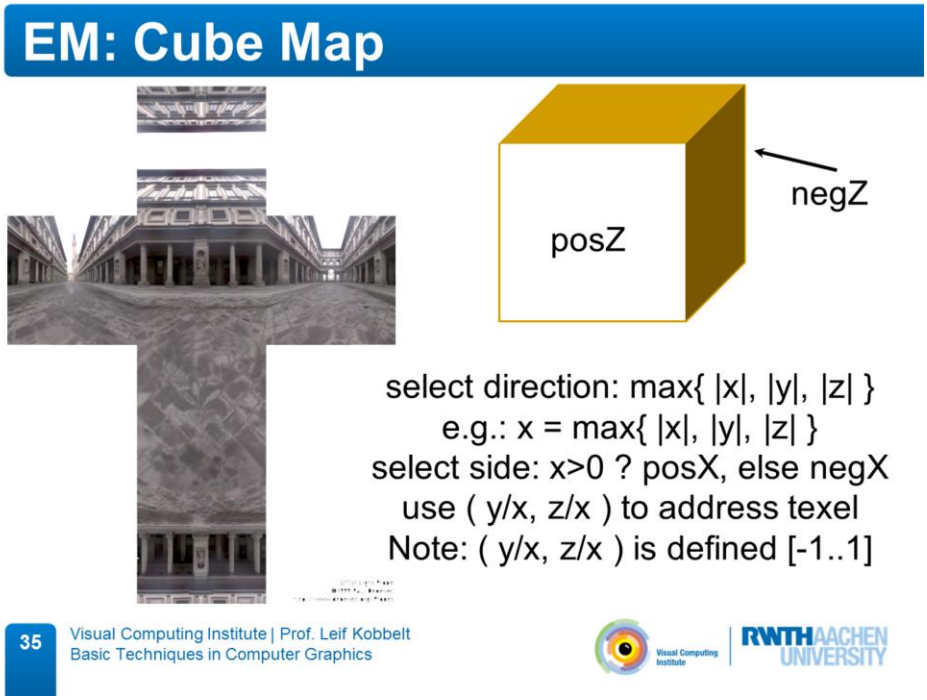
Sphere Maps are easy to create from real environments by taking a photo of a reflecting ball. But if you‘re talented you can also draw them by hand...

EM: Parabolic Mapping

- Idea: Use two images to map the whole environment



Compared to sphere maps a parabolic map can map the whole environment. It is more uniformly sampled than the other alternatives and only needs two textures compared to 6 for a cube map. However, the creation on standard hardware is not as trivial as for cube maps because like on sphere maps the environment gets „bend“ and straight lines will become curves which makes it necessary to tessellate the scene...



Cube maps have the advantage that they can be created in a rendering application itself by rendering the six faces with a field of view of 90 degrees.

When implemented on your own in a shader note that textures normally are defined from 0..1, so you have to scale the texture coordinates.

In some implementations the texture coordinate is defined as y/|x| , z/|x| - using textures defined in that way will appear flipped.

OpenGL already implements special cubemap textures and lookups into them by using the reflection vector directly!

EM: Cube Map

→ precompute lighting
(viewer and light source at infinity)

→ texel address is independent
from the length of the vector

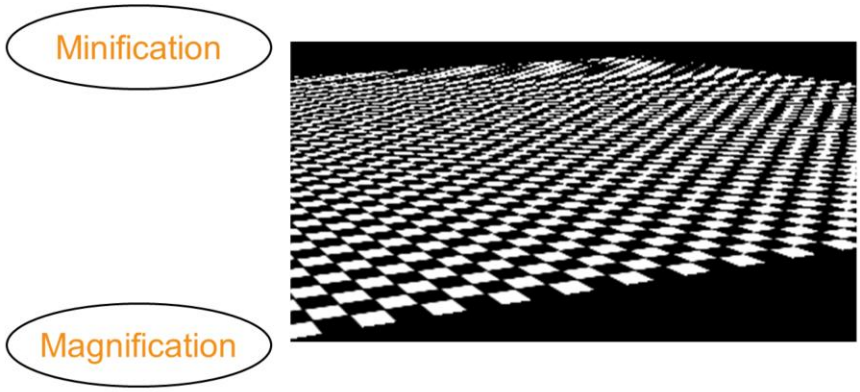
Environment Maps

- Mapping the real world?
  - Sphere Maps are easy to create
- Mapping a virtual scene?
  - Cube Maps are the easiest to render

Overview

- Texture Mapping
- **Texture Anti-Aliasing**
- Geometry Anti-Aliasing

Anti-Aliasing



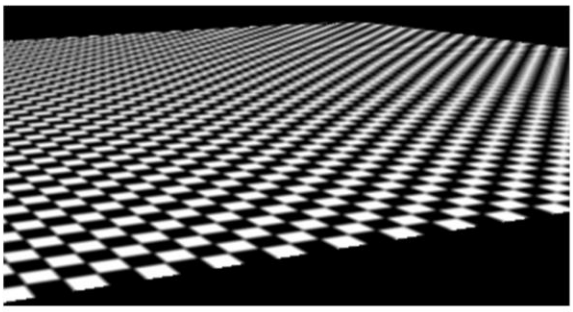
With point sampling we will will see different artifacts:

In areas with magnification we will see jaggies, individual Texels are visible with hard edges.

In areas with minification the point sampling can result in moire-like patterns. The higher the texture frequency, the more visible will the problems become.

Anti-Aliasing

Minification



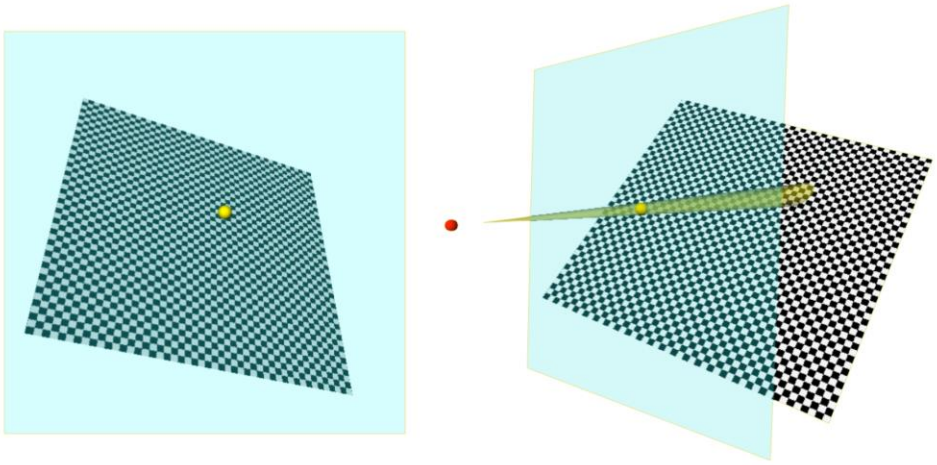
Magnification

The expected result.

Anti-Aliasing

- Point sampling is the wrong model !
  (Sampling density depends on parameterization)
- Integrate over pixel area

Anti-Aliasing

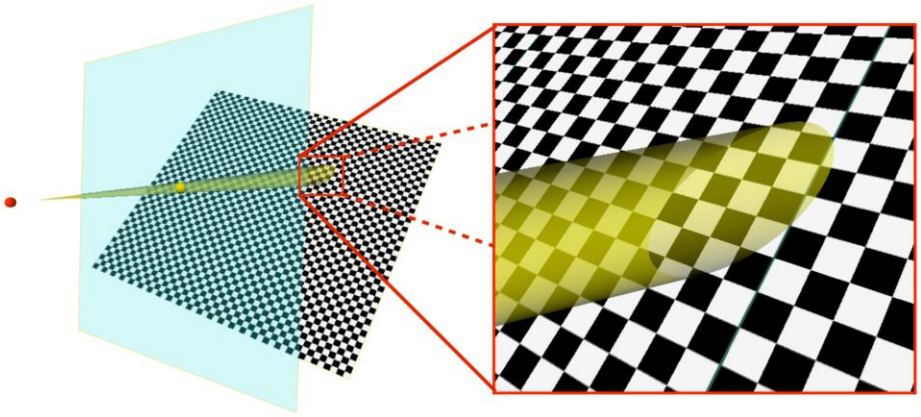


Left: the green dot shows what area should contribute to one pixel in the final image.

Right: that area gets projected from the camera (red) onto the geometry.

We can also think of a pixel as a square and the geometry as a pyramid.

Anti-Aliasing



In a close-up view we can see the area we have to integrate to calculate the correct final color value. With point sampling we would only look up the color at the center of that ellipse.

Depending of our view of the sampling that we want to do, this also could be a 4-sided polygon (when we think of mapping a pixel) instead of an ellipse (that could be our view while raytracing a beam).

Anti-Aliasing

- Point sampling is the wrong model !
  (Sampling density depends on parameterization)
- Integrate over pixel area
  (usually approximated by an ellipse)
- Supersampling
- Pseudo Anti-Aliasing by Mip-Mapping
  (store same texture in different resolutions)

The correct way would be to integrate over the pixel area, but that is usually to slow. With supersampling we would sample multiple points in that area and average the values. This can be very easily implemented in ray-tracing! For rasterization based rendering the common way for texture filtering is to use lower resolution textures while the object is far away. Mip-Maps are sets of different resolution versions of the same texture. The GPU can decide per pixel to render what Mip-Map level to query.

MIP-Mapping

- Introduced 1983
- MIP: „**M**ultum **I**n **P**arvo“ (lat)
  „many things in a small place“
- Increases texture size by 33%

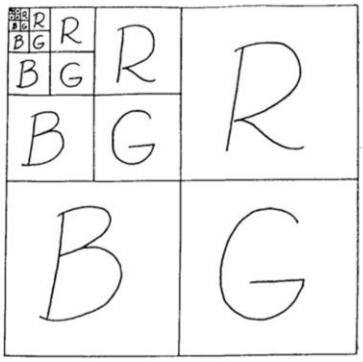
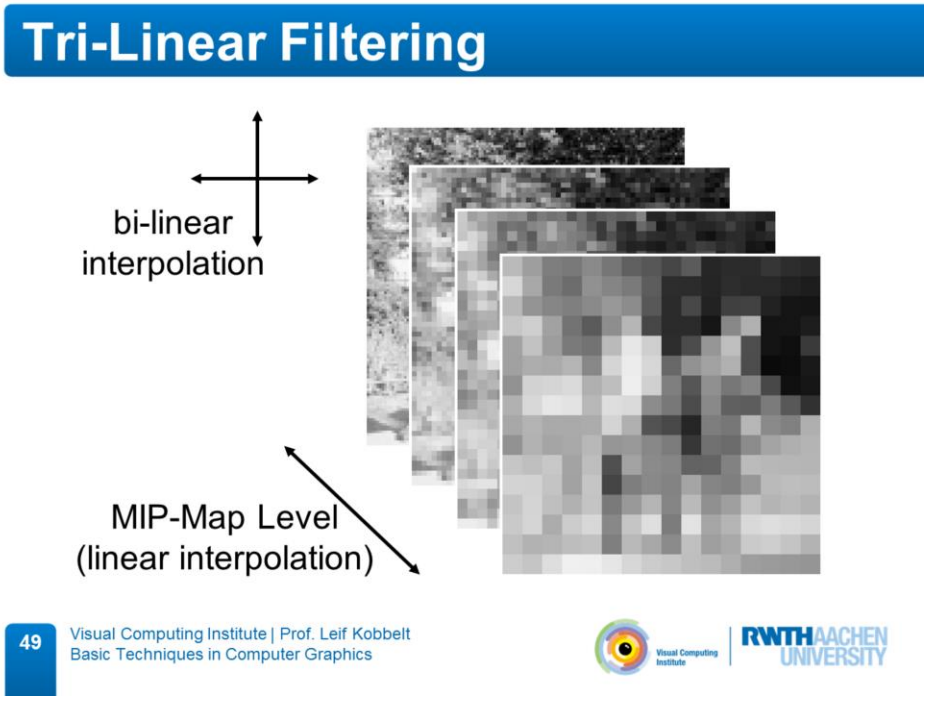


Figure (1)
Structure of a Color Mip Map
Smaller and smaller images diminish into the upper left corner of the map. Each of the images is averaged down from its larger predecessor.

Lance Williams: Pyramidal Parametrics (1983)

This downsampling is a rough approximation of the lowpass filter that would really be needed.



From the distance of an object we have to calculate the most appropriate Mip-Map level. Most of the time this level will be between two precalculated Mip-Map levels, that‘s why the resulting color gets linearly interpolated between these two levels.

In addition there will be a bi-linear interpolation within the levels which leads to tri-linear filtering.

Bi-Linear Interpolation

texture coordinates (u,v) = (i,j) + (u´,v´)

→ i,j : integer address for texels
→ $0 \leq u´,v´ < 1$

Texture Parametrization

usually $(u,v) \in [0,1)^2$

→ i = floor(u * width)
j = floor(v * height)
u´ = u * width - i
v´ = v * height - j

if $(u,v) \notin [0,1]^2$

→ u ← u mod 1.0
v ← v mod 1.0

„periodic“

Texture Parametrization

usually $(u,v) \in [0,1)^2$

→ i = floor(u * width)
j = floor(v * height)
u´ = u * width - i
v´ = v * height - j

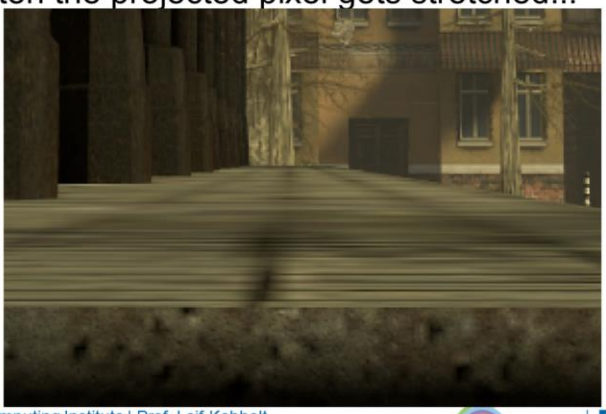
if $(u,v) \notin [0,1]^2$

→ u ← min(1.0, max(0.0, u))
v ← min(1.0, max(0.0, v))

„clamped“

Anisotropic Filtering

- Mip-Mapping provides an equal pre-computed filtering in x- and y-direction
- But often the projected pixel gets stretched...



The projected pixel onto the floor gets very long, so the ideal Mip-Map level in x- and y-direction differ. A projected pixel can span a few texels in the x-direction but dozens in y-direction!

Images from:
http://www.pcgameshardware.com/aid,743498/Geforce-GTX-480-and-GTX-470-reviewed-Fermi-performance-benchmarks/Reviews/?page=3

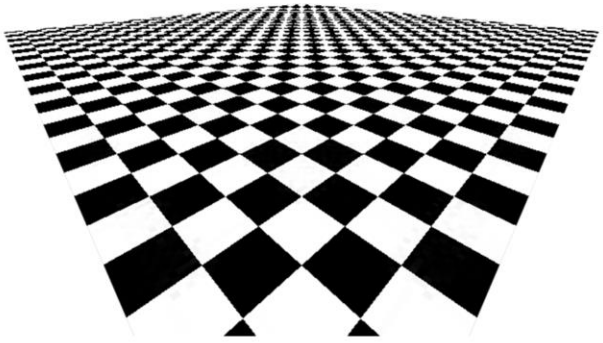
Anisotropic Filtering

- Solution: Sample multiple texture positions
- Direction is based on the projected pixels shape

Current hardware can sample up to 16 positions in the projected shape from the Mip-Maps. Each sampling will result in a tri-linear interpolation. For each tri-linear interpolation four texture samples for two mip-map levels are required. A total of 128 texture reads will be needed for AF with 16 samples as shown in the image. This is a worst-case scenario as the GPU can decide how many samples it will query based on the angle a texture is looked at.

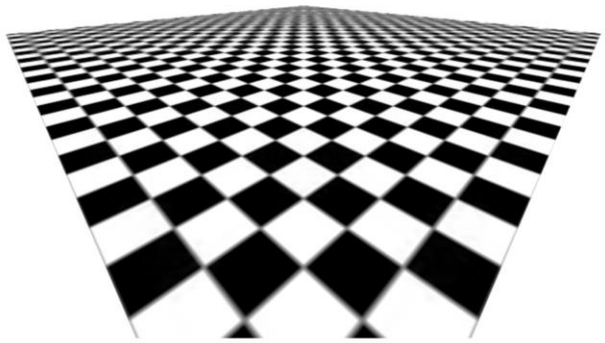
Images from:
http://www.pcgameshardware.com/aid,743498/Geforce-GTX-480-and-GTX-470-reviewed-Fermi-performance-benchmarks/Reviews/?page=3

Point Sampling

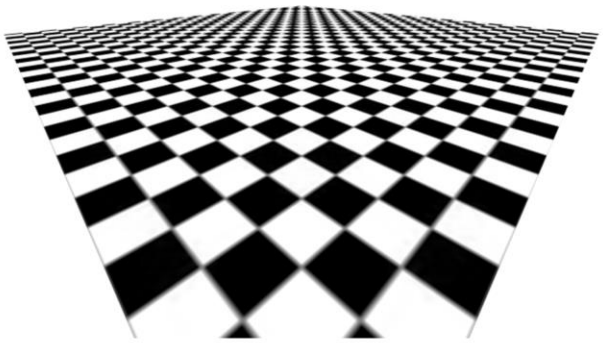


A short recap of the texture filtering modes

Tri-Linear Sampling



Anisotropic Sampling



Anisotropic filtering is an OpenGL extension thats very common. The maximum value of samples that can be used for filtering is implementation dependent, here 16 is just a (high) example.
Anisotropic filtering has to be combined with the Min and Max filtering options and is no replacement.

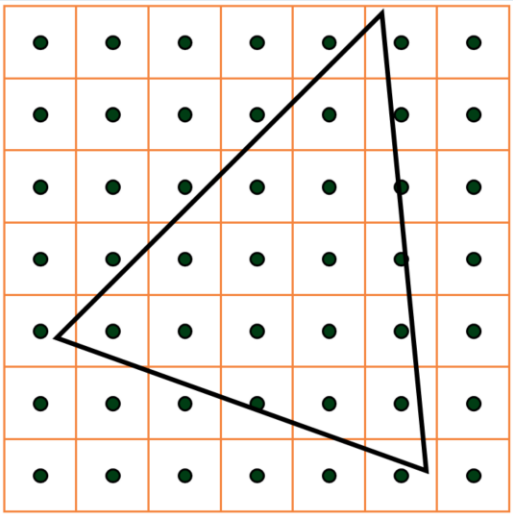
Textures

Demo

Overview

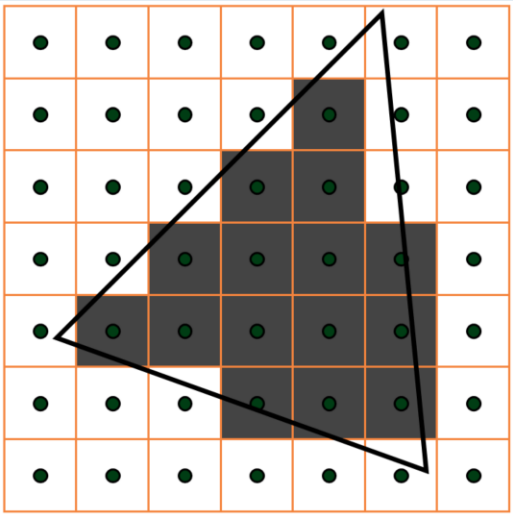
- Texture Mapping
- Texture Anti-Aliasing
- **Geometry Anti-Aliasing**

Anti-Aliasing II



Aliasing does not only occur inside of polygons while adding the texture, but also at the edges of polygons.

Anti-Aliasing II



A fragment gets created if the triangle covers the center of the resulting pixel. This is called the sample position. Again the correct way would be to integrate over the final pixel area to measure the triangles contribution to this pixel.

The same problem occurs with ray-tracing: think of the sampling positions as rays: we will get the same hard edges.

Full-Scene AA: FSAA

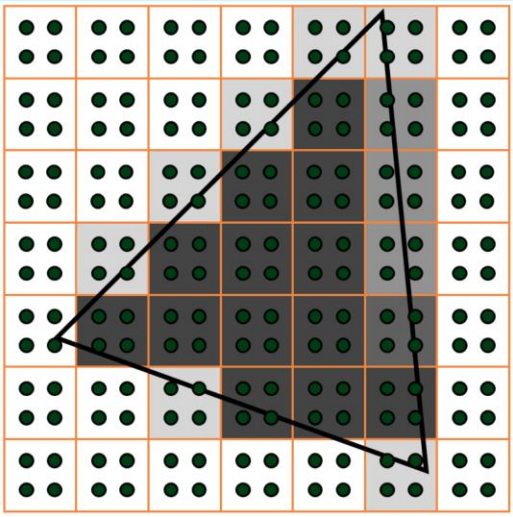
„trivial“ implementation:

rasterization:

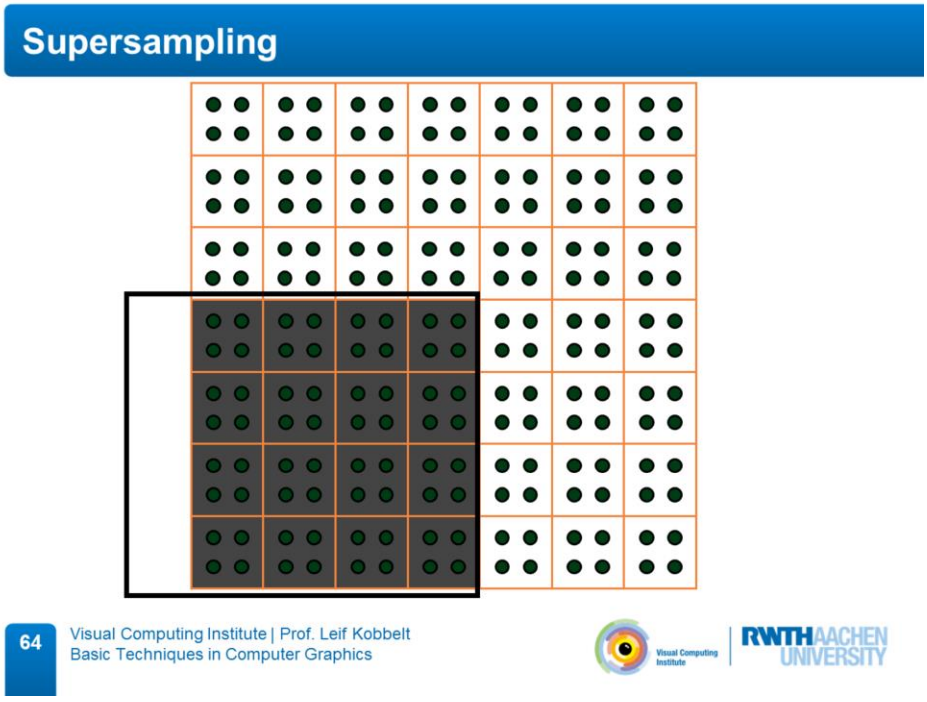
- render at twice the size
- scale down the resulting image

Rendering internally in a higher resolution and downscaling the result is trivial to implement but slows down rendering speed and also needs much more video memory for the renderbuffer and depthbuffer. Instead of just increasing the resolution in each direction by a factor of two, any other integer factors are possible as well!)

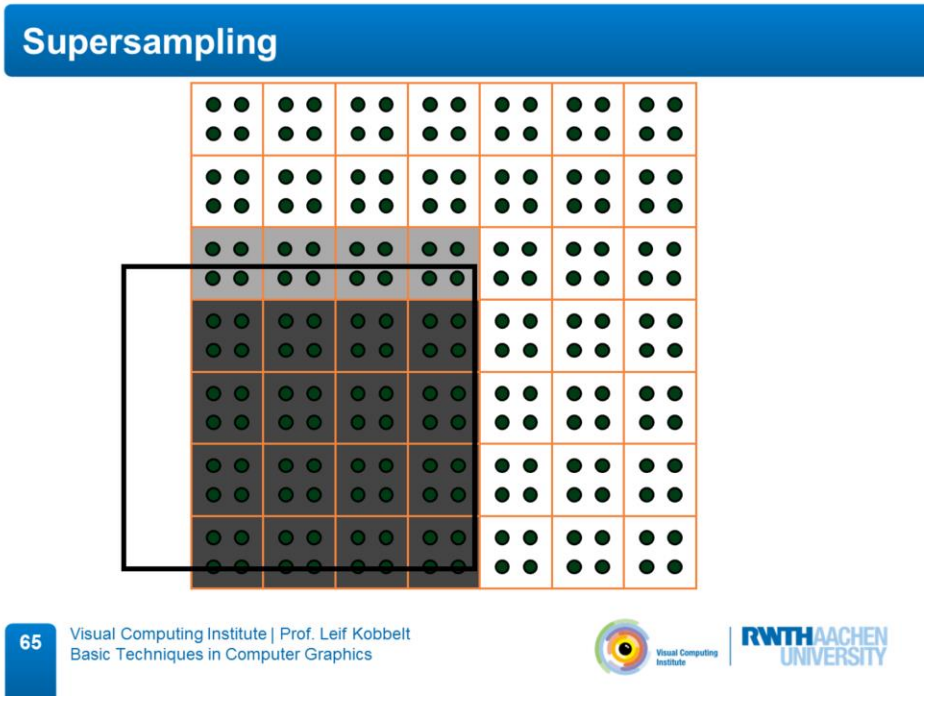
Supersampling



Instead of integrating, multiple samples within the pixel area get evaluated to check how large the coverage of the triangle will be. If at least one sampling point lies within the triangle, a fragment gets created.
Supersampling in general means computing more than one sample per pixel.

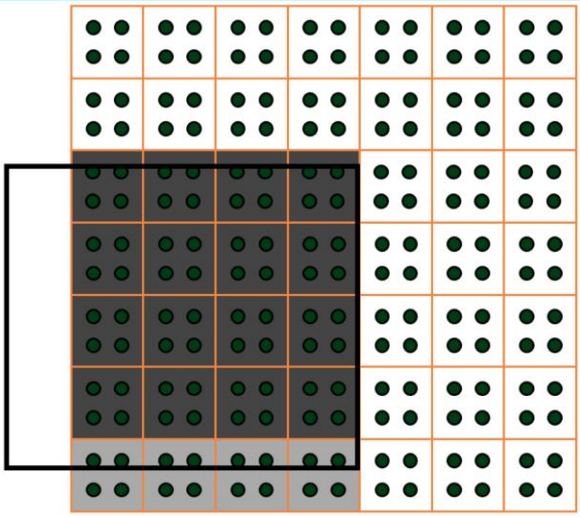


The regular pattern of supersampling has the disadvantage that

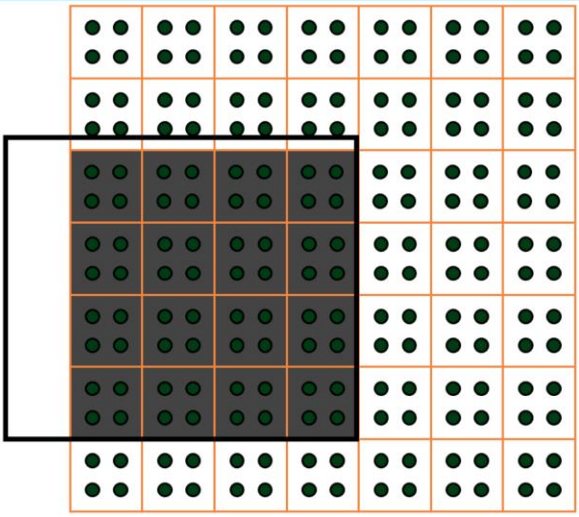


Only one shade of gray is created even tho 4 samples are used. This happens for all horizontal and vertical edges which are quite common when rendering artificial structures (e.g. buildings from street level).

Supersampling



Supersampling



Full-Scene AA: FSAA

- For each pixel of the final image
  - Check at N positions if the triangle covers this area
  - Compute the color N times
  - Store N color & depth values

Rendering the image in twice the size will result in a regular 2x2 sampling grid with N=4, but there are other patterns with more or less sampling positions possible.
Computing the color N times means running the fragment shader N times which will result in N very similar colors.

Multisample AA: MSAA

- For each pixel of the final image
  - Check at N positions if the triangle covers this area
  - Compute the color once
  - Store N color & depth values

Multisampling saves some computations by only calculating the color (=running a fragment shader) once for all covered samples. This is a bit less accurate but mostly not noticeable.

Multisample AA: MSAA

Sampling pattern:

- Red: positional sample
- Green: shading sample

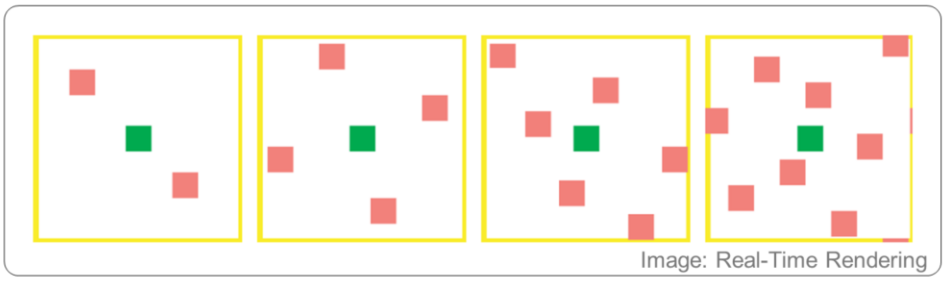


Image: Real-Time Rendering

If all red positional samples are covered, the color gets calculated at the green shading sample in the middle.
If the green sample itself is not covered, the shading position can get shifted. The exact shifting behavior can be influenced by the fragment shader.
The more “chaotic” sampling patterns help to reduce artifacts when rendering horizontal and vertical edges. For most patterns an edge can be found which also would not create all possible shades of coverage, however, those edge orientations are less likely to occur than 90 or 45 degree edges!
Remember that with the rasterization algorithm from Pineda the test whether a fragment is inside of a triangle can be performed at any position independently! This helps supporting arbitrary sampling positions.

Anti-Aliasing: Sampling Patterns

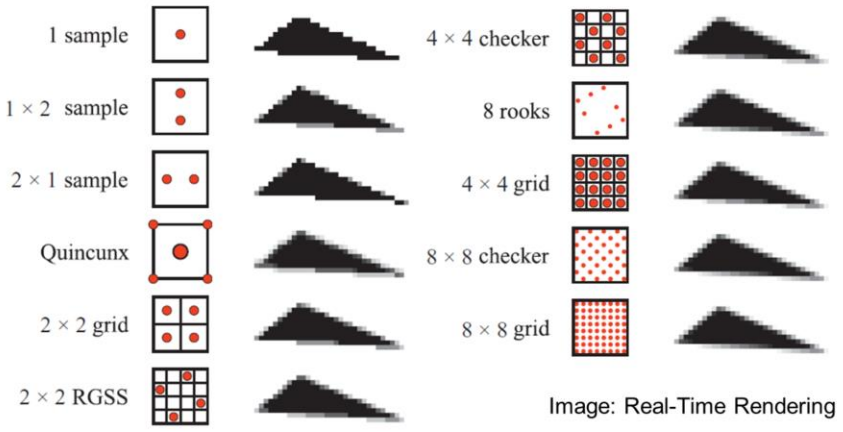


Image: Real-Time Rendering

Different sampling schemes.

RGSS: Rotated Grid Supersampling

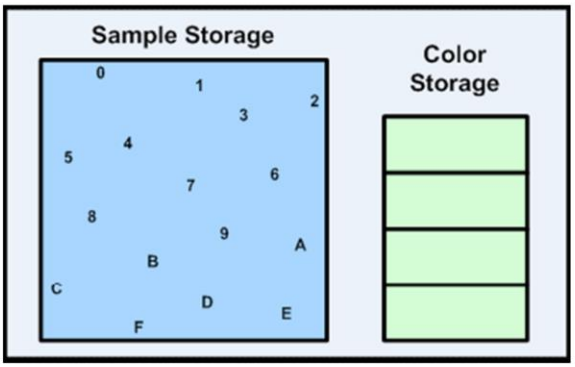
On some modern hardware (2014), the sampling pattern can get defined by the application.

Coverage Sampling AA: CSAA

- For each pixel of the final image
  - Check at N positions if the triangle covers this area
  - Compute the color once
  - Store up to M color & depth values
    - Each of the N sample positions will point to one of the M color/depth pairs

For 8x MSAA 8 color/depth pairs have to be stored, because the color was computed only at one position for each polygon anyway these values don‘t differ much. In fact it‘s rare to have a pixel where 8 different polygons contributed some color.
The idea behind coverage sampling is to store less distinct color values and the coverage of the contributing polygon.

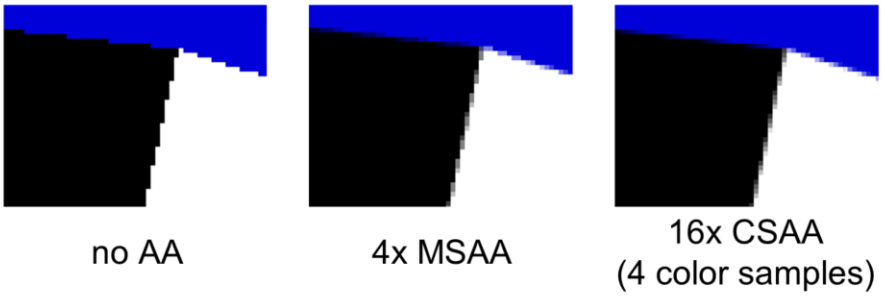
Coverage Sampling AA: CSAA



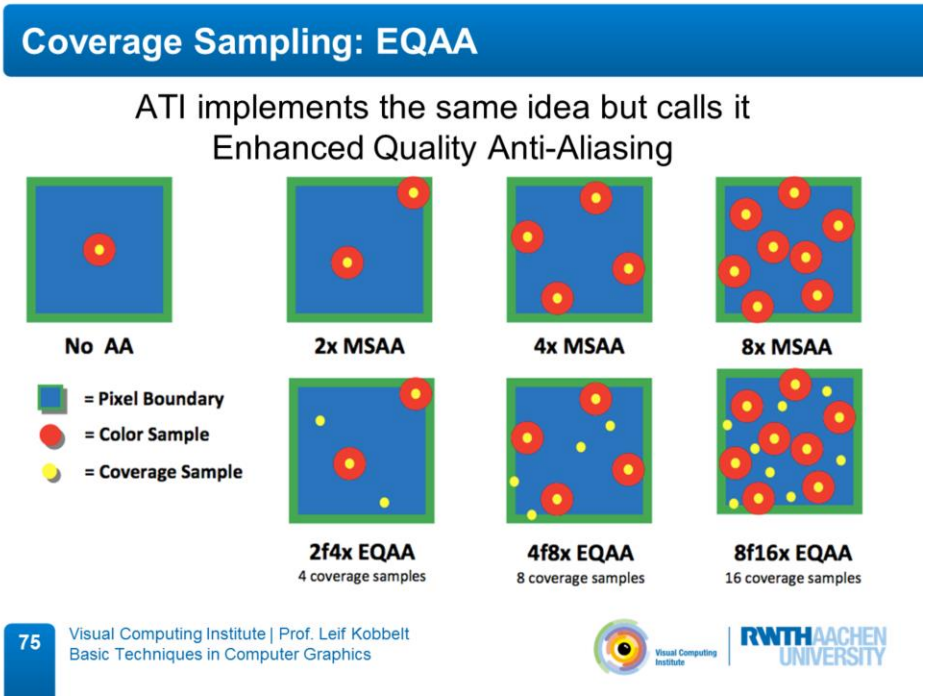
For each sample a 2 bit index into the color storage is stored.

For example 16x CSAA on NVidia GPUs store 4 color&depth values but 16 coverage samples.
Artifacts will occur if more triangles contribute to one pixel than there are entries in the color storage.
Sample storage in this example: 16*2bit = 32bit = 4byte per pixel.
One color storage entry in this example: 24bit depth + 8bit stencil + 4*8bit color = 64bit = 8byte per sample.
The sample storage has only half the size of one additional color sample!
(Image: http://developer.nvidia.com/csaa-coverage-sampling-antialiasing)

Coverage Sampling AA: CSAA



(Image examples: http://developer.nvidia.com/csaa-coverage-sampling-antialiasing)



The basic idea to test against more coverage samples than the number of actually saved color values is the same for NVidias CSAA and ATIs EQAA.

Image: http://developer.amd.com/sdks/radeon/assets/EQAA%20Modes%20for%20AMD%20HD%206900%20Series%20Cards.pdf

Anti-Aliasing II

- CSAA is only one example of improving existing AA techniques
- Too many variants to cover them all here!

FSAA with SSAA is mostly never used in real-time applications like games, MSAA is a quite common trade off. Alternatives are CSAA as well as methods that handle transparent objects better and edge detecting anti-alias schemes.

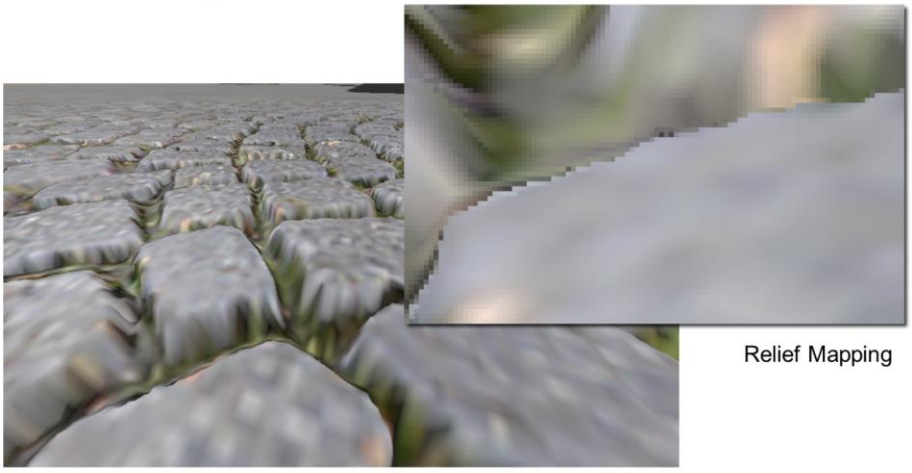
Multisampling Limitations

- Supersampling (SSAA, FSAA):
  - very high computational cost
  - very high memory cost
  - good result (given enough samples)
- Multisampling (MSAA, CSAA, EQAA, etc.):
  - high computational cost
  - very high memory cost
  - good results on edges

The two classes of AA summed up. MSAA in all variants is faster but needs a lot of memory. The downside is, that it only works well on geometric edges.

Multisampling Limitations

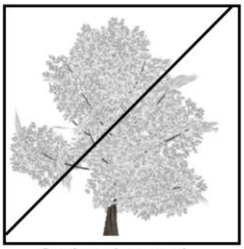
Aliasing introduced by the Fragment Shader



Relief Mapping

Multisampling Limitations

Aliasing introduced by the Fragment Shader



2 triangles and a texture

detail view

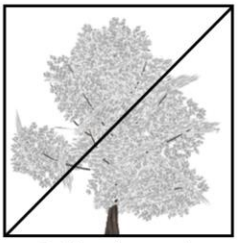
rendering

if alpha < 0.5: colorize
else: discard fragment

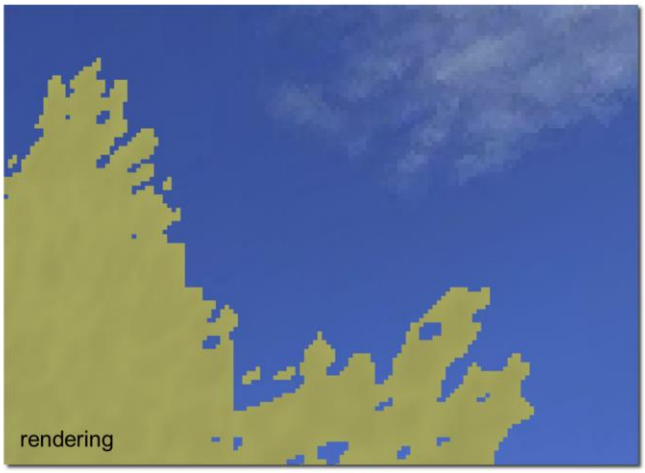
Another example of aliasing which gets created in the fragment shader: A tree gets rendered as a quad (two triangles) with a texture (shown is the alpha channel with white being transparent in this case). To avoid sorting this partially transparent object it is not blended into the framebuffer but the fragment shader discards all fragments where the alpha is above a given threshold. This introduces aliasing which can not be handled by MSAA as most fragments are inside of the original quad and thus the fragment shader will only be called once!

Multisampling Limitations

Aliasing introduced by the Fragment Shader



2 triangles and
a texture



Detail view to see the aliasing.

Postprocessing Anti-Aliasing

General idea:

- Render without AA into a texture
- Detect edges
- Blur edges

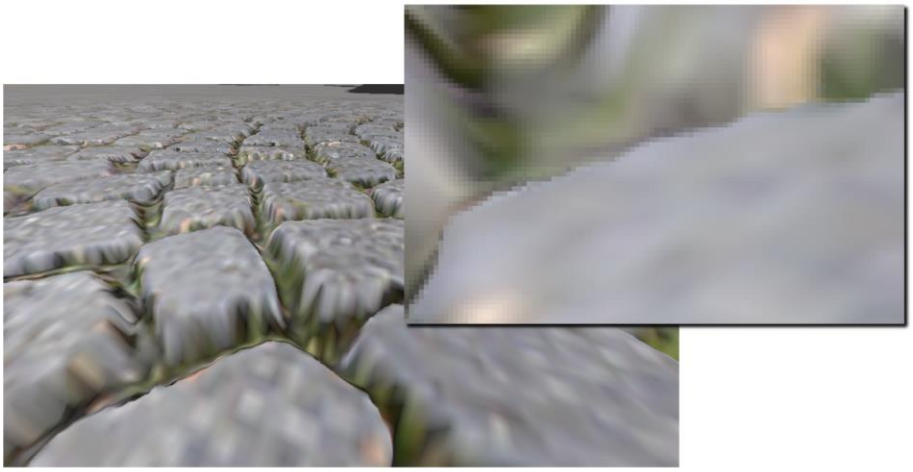
The edge detection can be performed on the color buffer, the depth buffer or any additional information from the rendering dependent on the technique used. Some newer algorithms even take the previous frame into account. Some techniques add one pass (e.g. FXAA), some multiple passes (3 in case of “Practical Morphological Antialiasing” by Jimenez et al., GPU Pro 2).

Postprocessing Anti-Aliasing

- Independent of scene complexity
- Dependent on resolution
- Predictable performance impact
- AA is performed everywhere

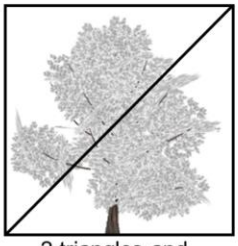
Postprocessing AA is independent on the geometric complexity of the scene as well as independent on the shader complexity. Thus the performance impact is predictable and only dependent on the resolution of the rendering. Dependent on the technique used no additional buffers/textures are needed.
AA is performed everywhere, even for procedural textures and inside regular textures - this might not always be preferred.
This also results in all hard edges being blurred, in some cases this is not wanted, e.g. blurring the edges between the scene and a UI / head up display. In those cases the UI should be drawn after the scene was processed with the AA algorithm.

FXAA



FXAA is one prominent example of these class of AA techniques.

FXAA



2 triangles and
a texture