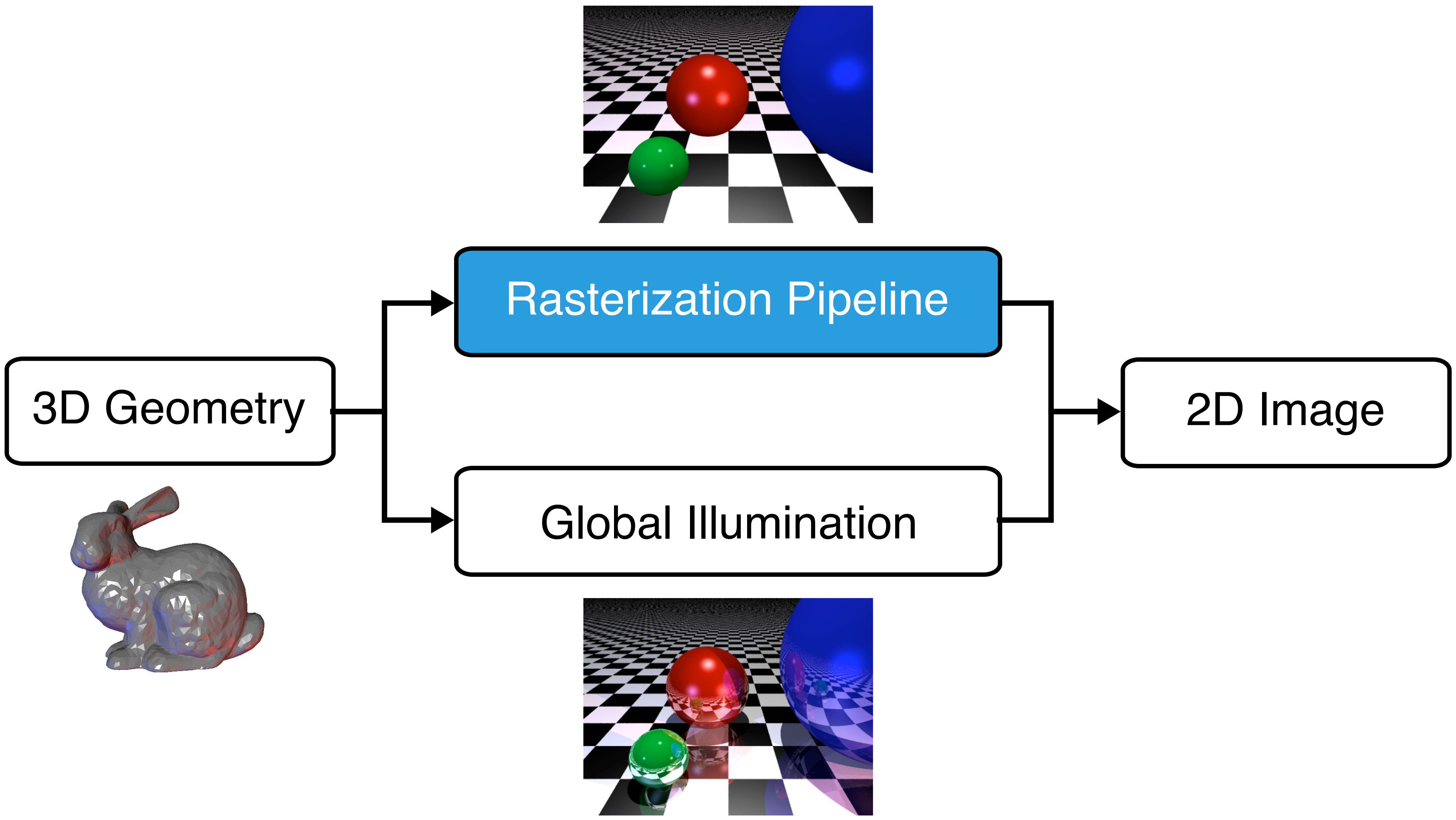
Computer Graphics

Advanced Rendering Methods

Mark Pauly

Geometric Computing Laboratory

Rasterization Pipeline



Quiz: Rasterization Pipeline

Which of the (incomplete) steps in the pipelines below are in the correct order?

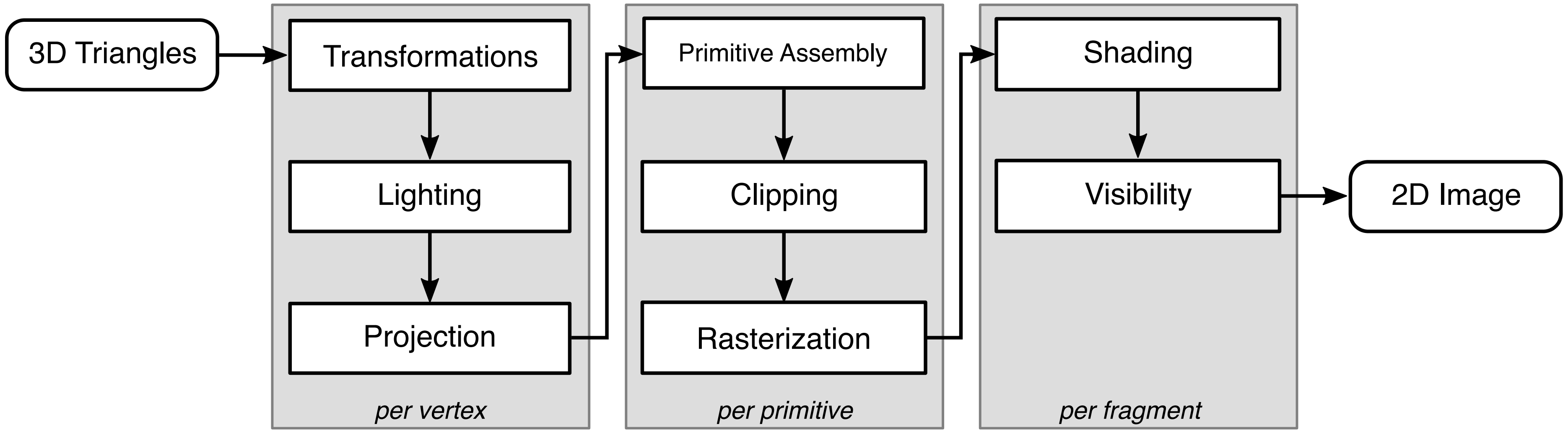
A: Transformations $\rightarrow$ Shading $\rightarrow$ Projection $\rightarrow$ Visibility

B: Lighting $\rightarrow$ Clipping $\rightarrow$ Rasterization $\rightarrow$ Visibility

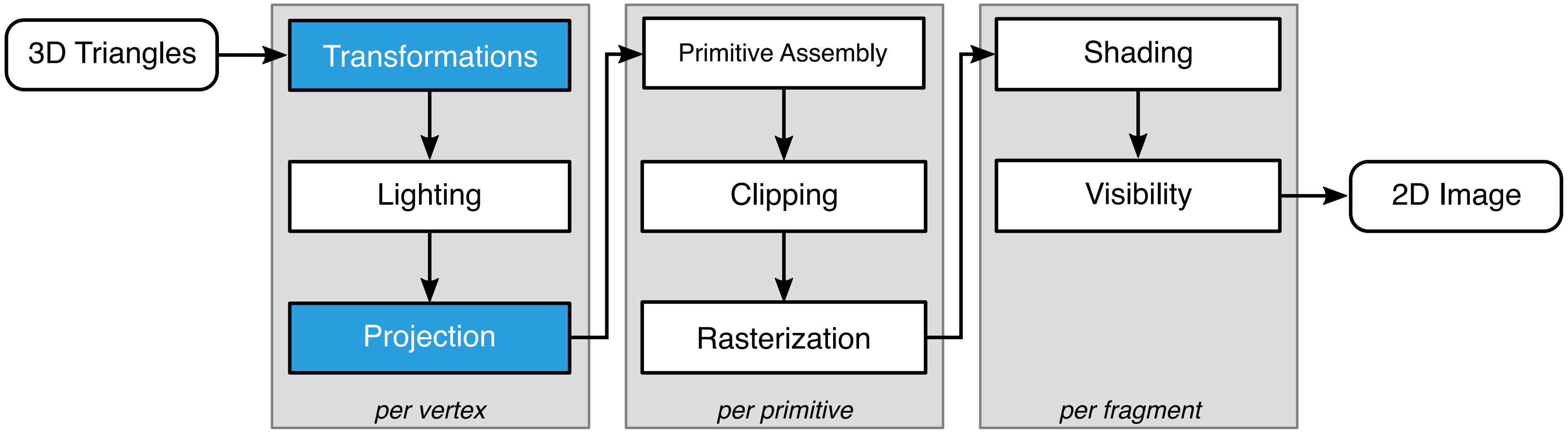
C: Projection $\rightarrow$ Rasterization $\rightarrow$ Clipping $\rightarrow$ Shading

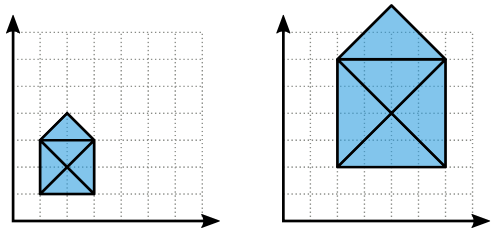
D: Transformations $\rightarrow$ Lighting $\rightarrow$ Rasterization $\rightarrow$ Projection

GPU Rasterization Pipeline

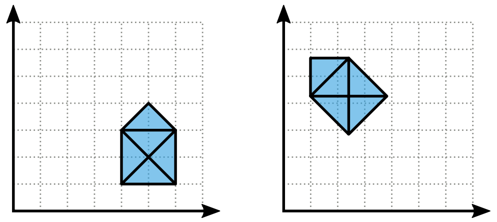


Transformations & Projections

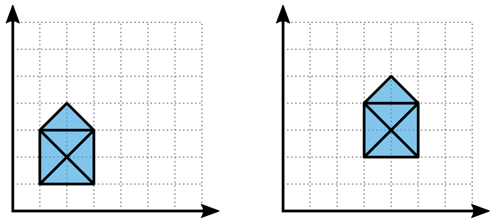




scaling

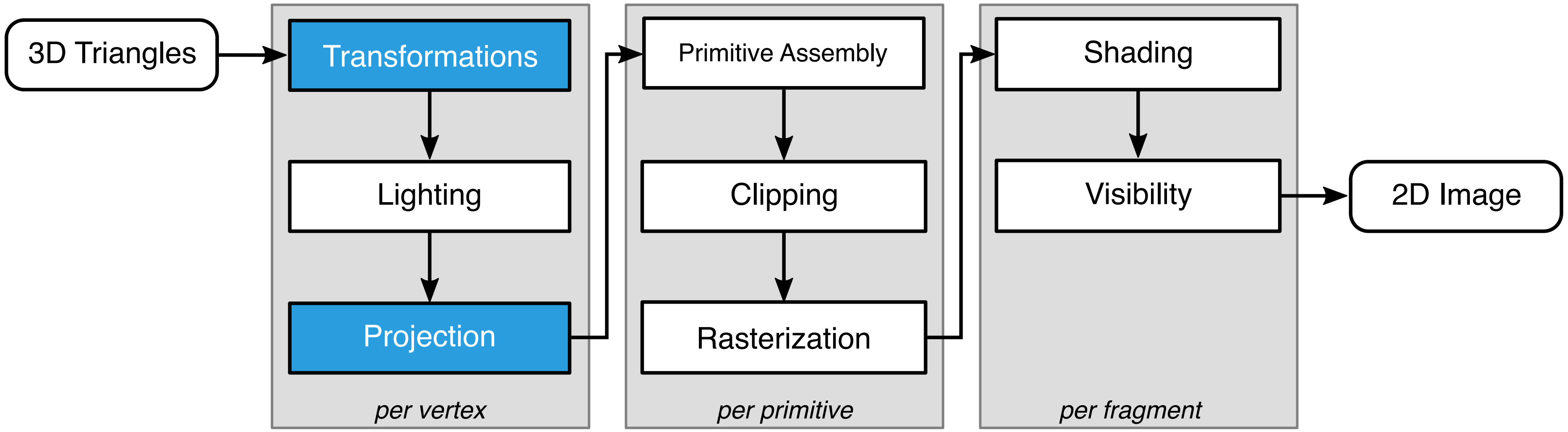


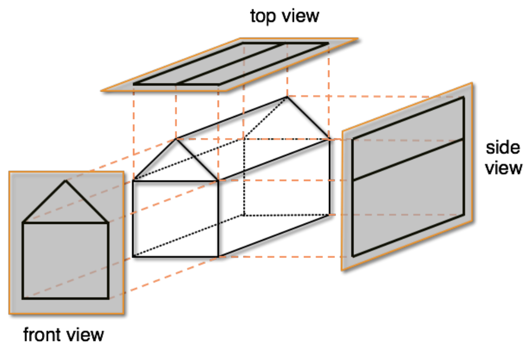
rotation

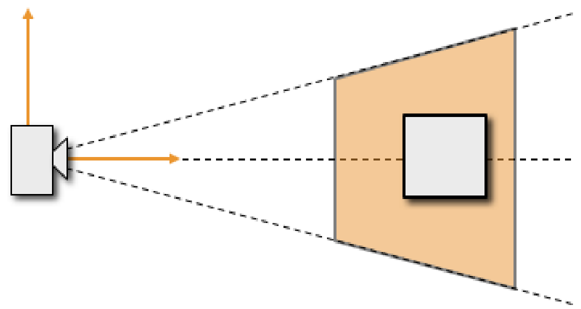


translation

Transformations & Projections

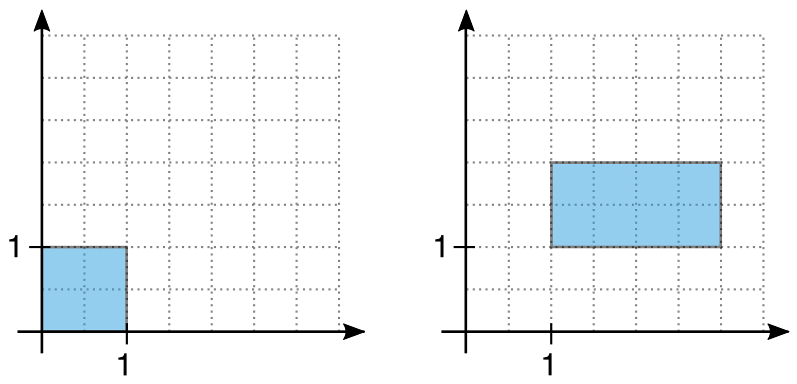






Quiz: Transformations

Which matrix computes the transformation on the right?



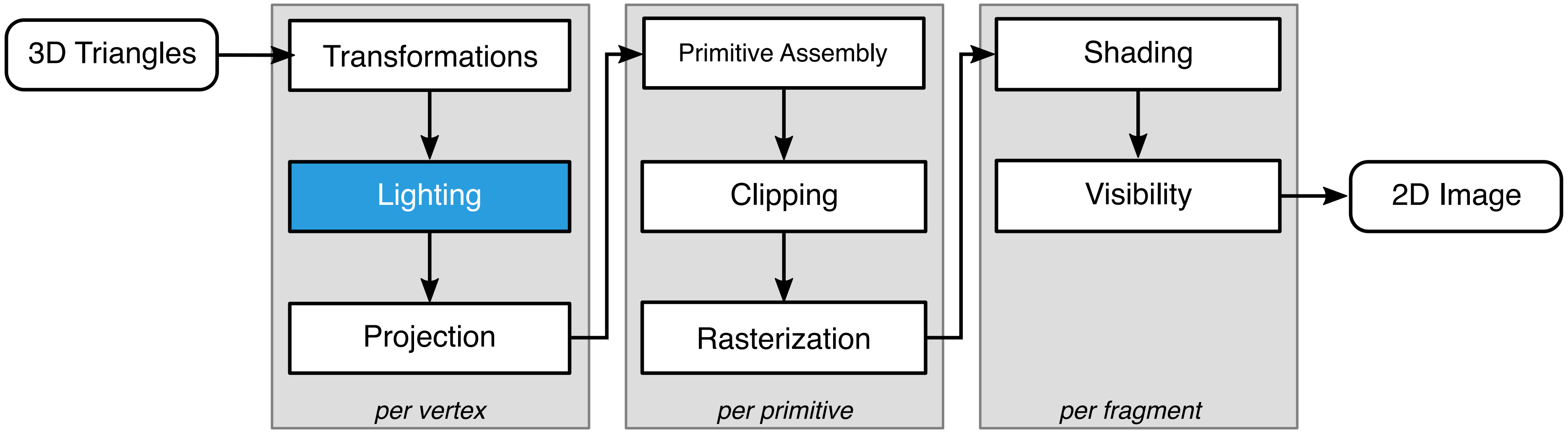
A: $\begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

B: $\begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 2 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 0 & 1 & 0 \\ -1 & 0 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

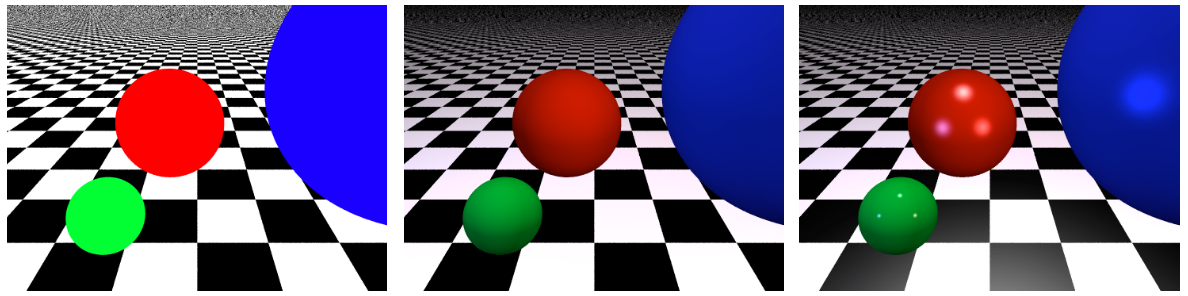
C: $\begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}$

D: $\begin{pmatrix} 2 & 0 & 1 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 1 \\ 0 & 0 & 1 \end{pmatrix}$

Lighting



Phong Lighting Model



Quiz: Lighting

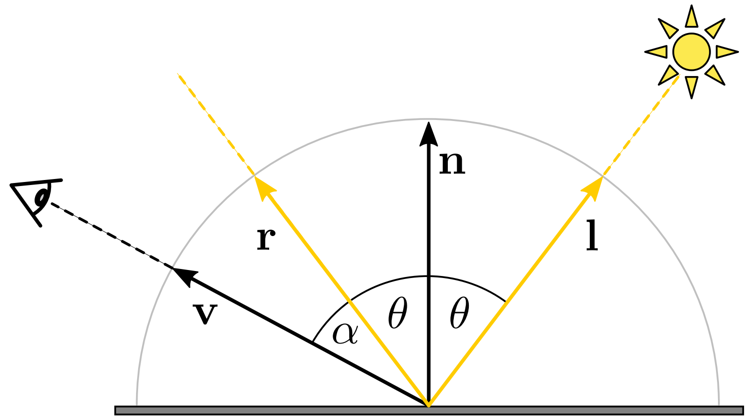
What is the specular component of Phong lighting?

A: $I_l m_s (\mathbf{r} \cdot \mathbf{v})^s$

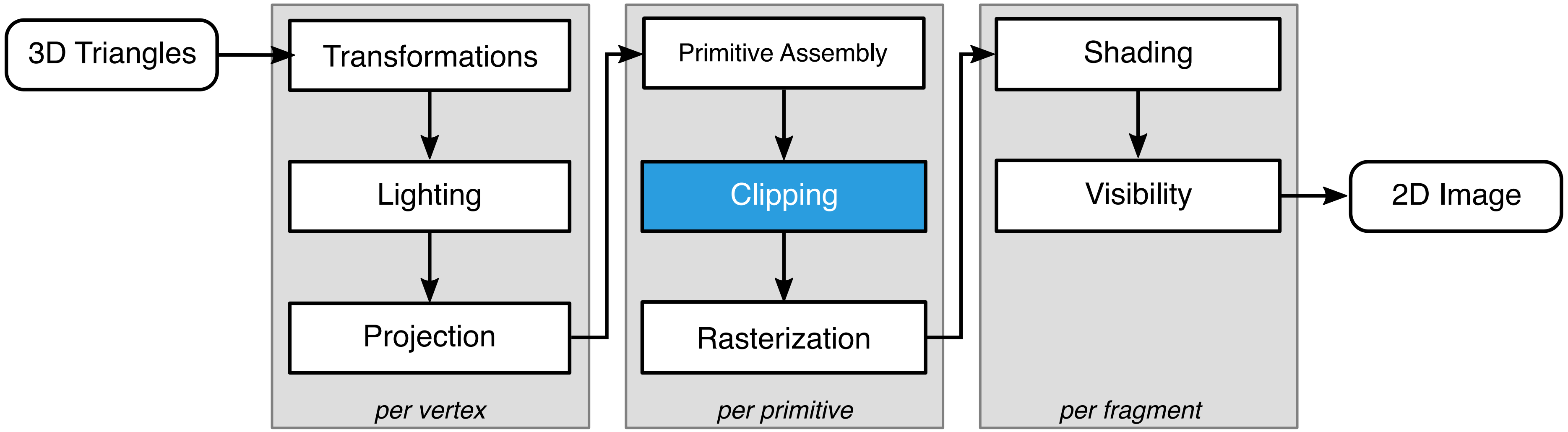
B: $I_l m_s (\mathbf{v} \cdot \mathbf{l})^s$

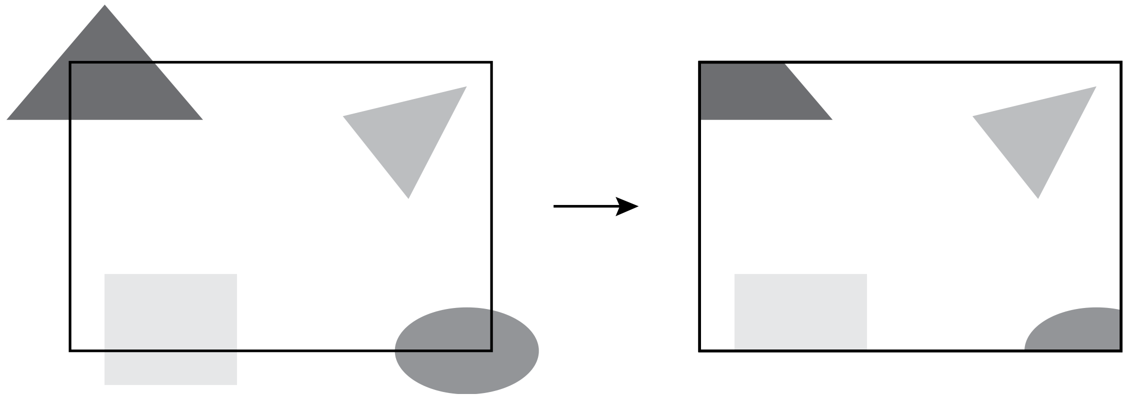
C: $I_l m_s (\mathbf{n} \cdot \mathbf{l})^s$

D: $I_l m_s (\mathbf{r} \cdot \mathbf{n})^s$

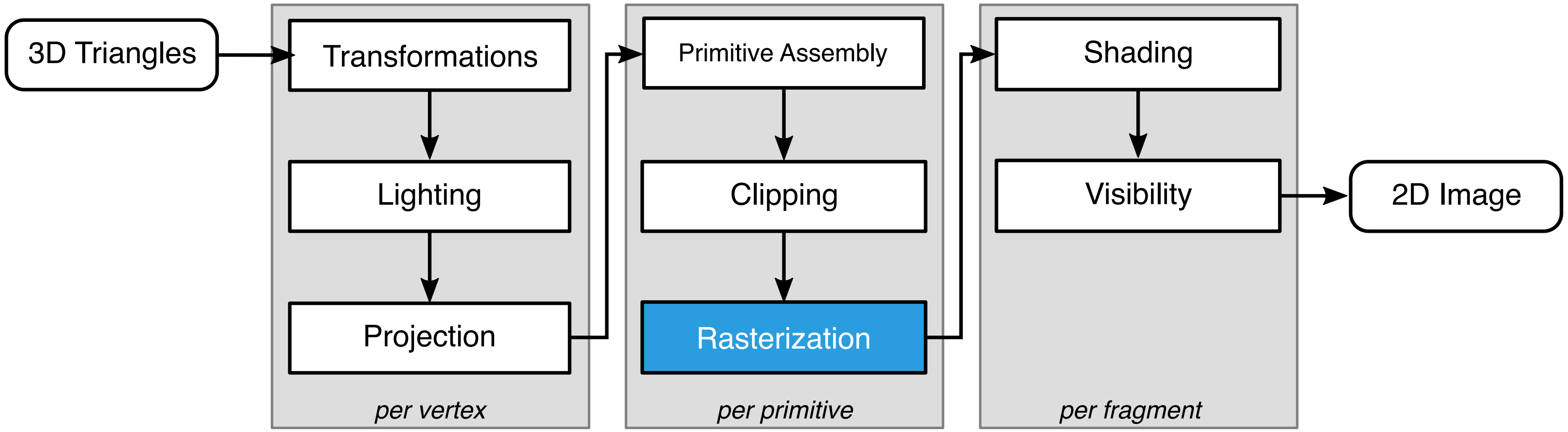


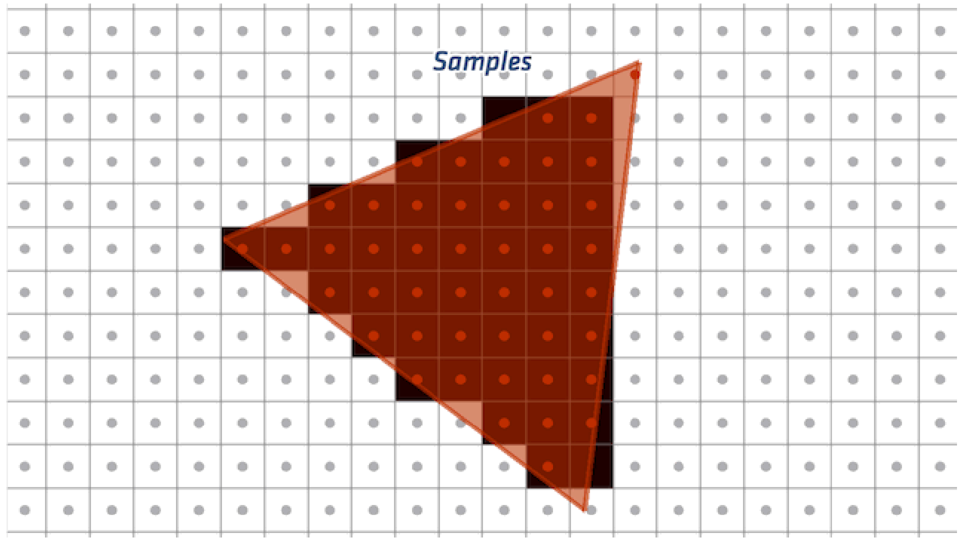
Clipping


3D Triangles
Transformations
Lighting
Projection
per vertex
Primitive Assembly
Clipping
Rasterization
per primitive
Shading
Visibility
per fragment
2D Image



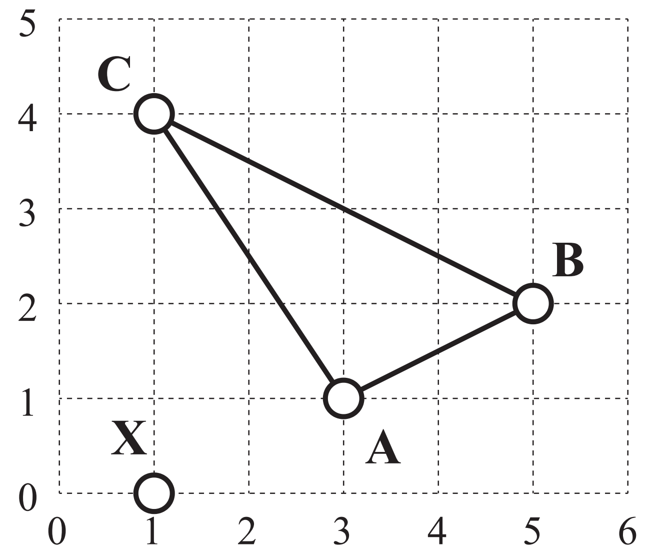
Rasterization




Samples

Quiz: Barycentric Coordinates

A point $\mathbf{X}$ is represented with respect to the vertices of a triangle $\mathbf{ABC}$ as $\mathbf{X} = \alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}$ with $\alpha + \beta + \gamma = 1$. Which are the correct coordinates for the point $\mathbf{X}$ shown in the figure?



A: $(1, -1, 1)$

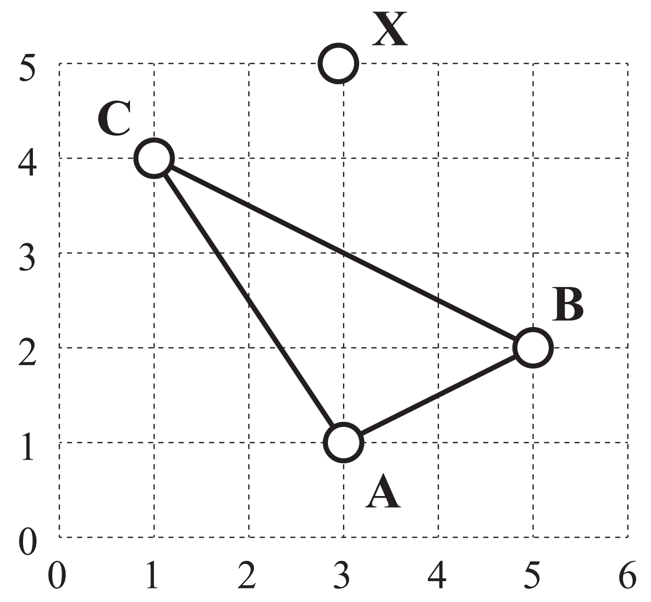
B: $(-1, 1, 1)$

C: $(-1, 2, 0)$

D: $(2, -1, 0)$

Quiz: Barycentric Coordinates

A point $\mathbf{X}$ is represented with respect to the vertices of a triangle $\mathbf{ABC}$ as $\mathbf{X} = \alpha\mathbf{A} + \beta\mathbf{B} + \gamma\mathbf{C}$ with $\alpha + \beta + \gamma = 1$. Which are the correct coordinates for the point $\mathbf{X}$ shown in the figure?



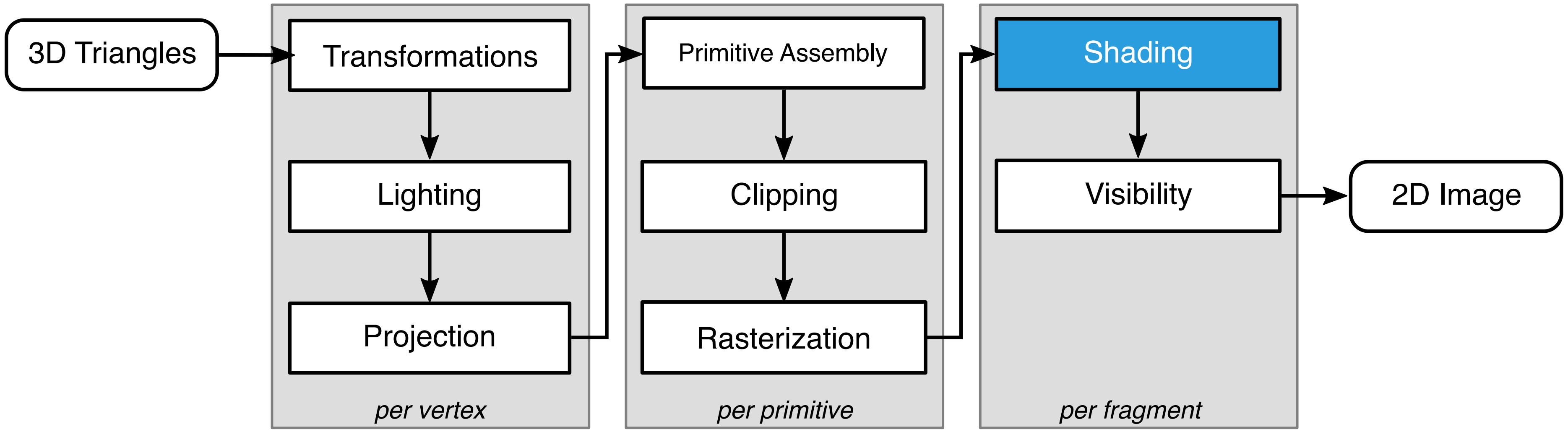
A: $(1, 1, -1)$

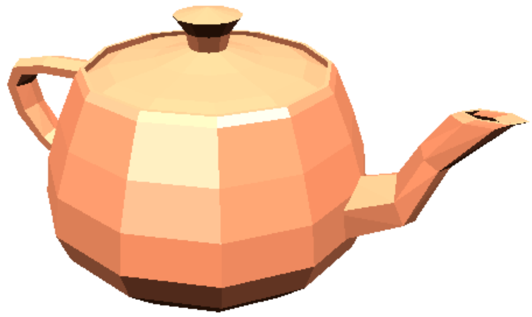
B: $(-1, 1, 1)$

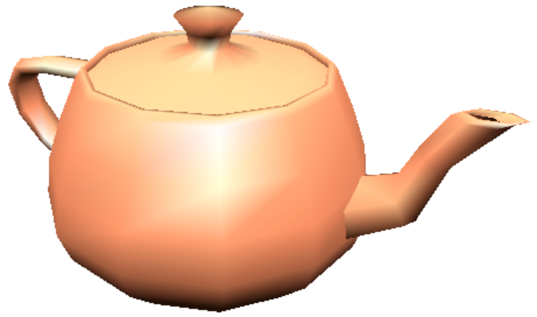
C: $(.5, 1, .5)$

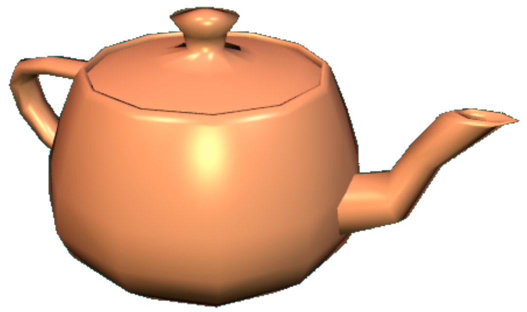
D: $(3, -1, -1)$

Shading

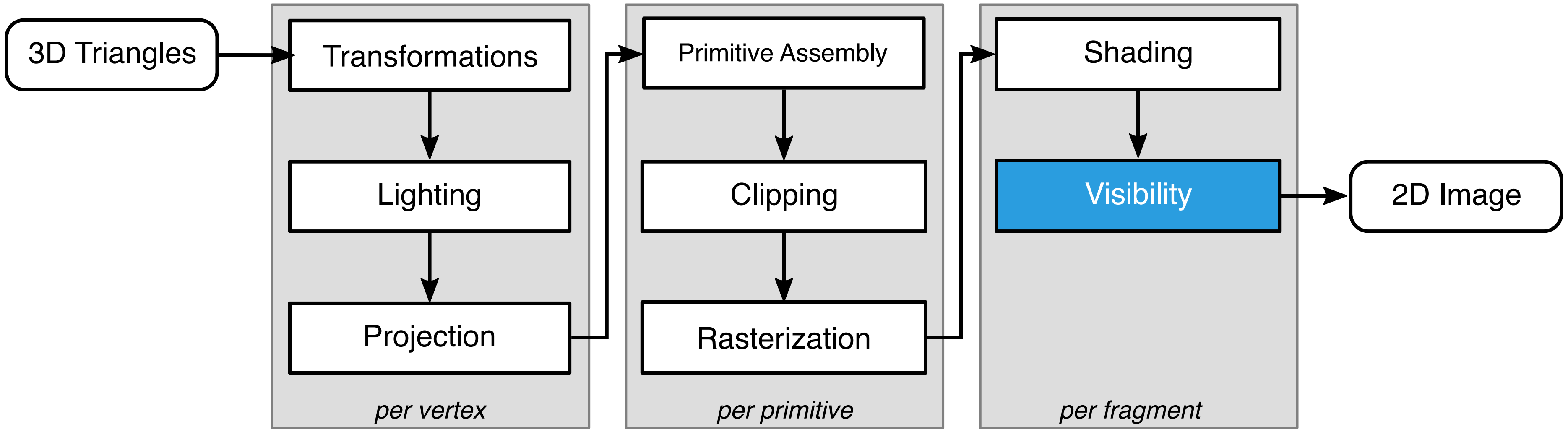


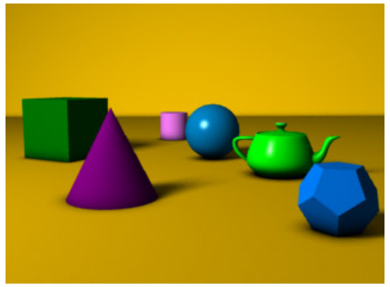

flat (constant)

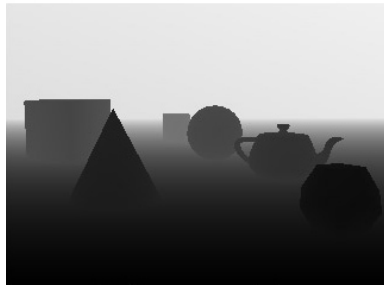

per-vertex (Gouraud)

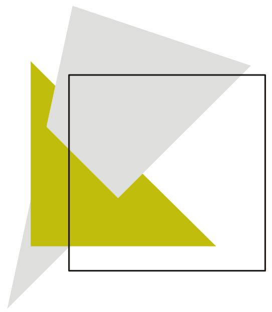

per-fragment (Phong)

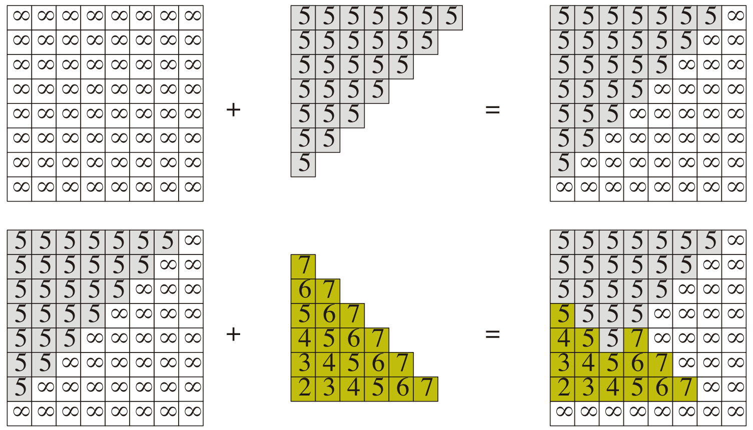
Visibility


3D Triangles
Transformations
Lighting
Projection
per vertex
Primitive Assembly
Clipping
Rasterization
per primitive
Shading
Visibility
per fragment
2D Image



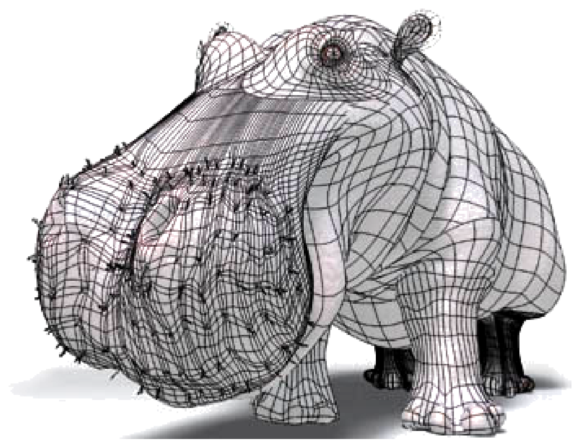


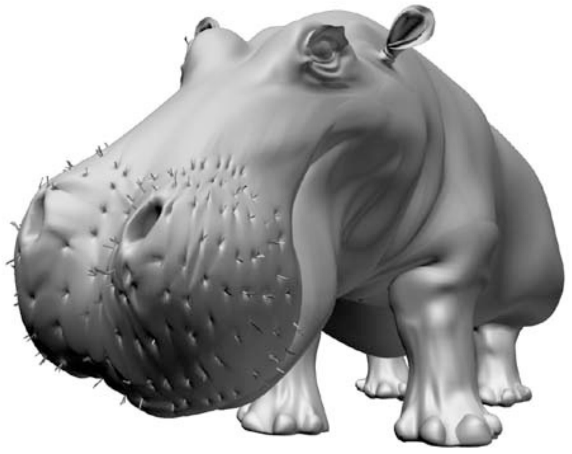


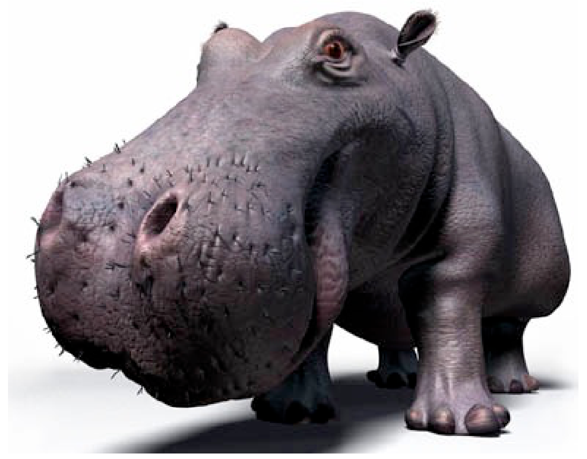


Materials & Texture

- Textures add visual detail without raising geometric complexity
- Textures allow us to model many surface properties:
  - material (diffuse + specular colors/coefficients), normal vector (normal mapping, bump mapping), geometry (displacement mapping), opacity (alpha mapping), reflection/illumination (environment mapping)


Geometry


+Lighting


+Texture

Images from http://www.3drender.com/jbirn/productions.html

Quiz: Textures

Which of the following statements is true?

A: Bilinear texture filtering eliminates aliasing.

B: There can be no mapping from the half-sphere to the plane that preserves area everywhere.

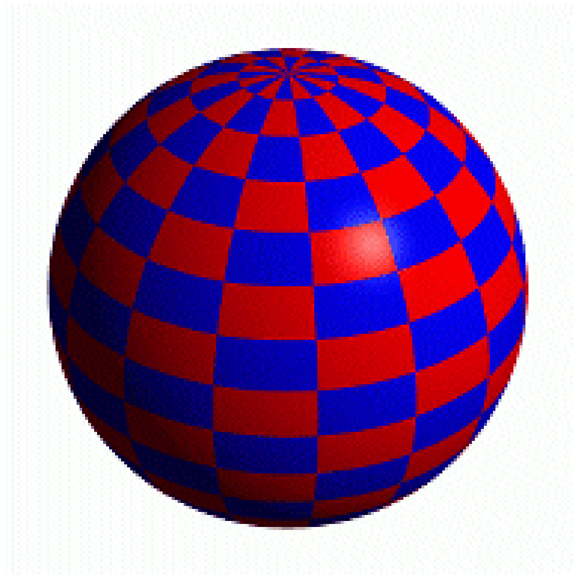
C: There can be no mapping from the half-sphere to the plane that preserves length everywhere.

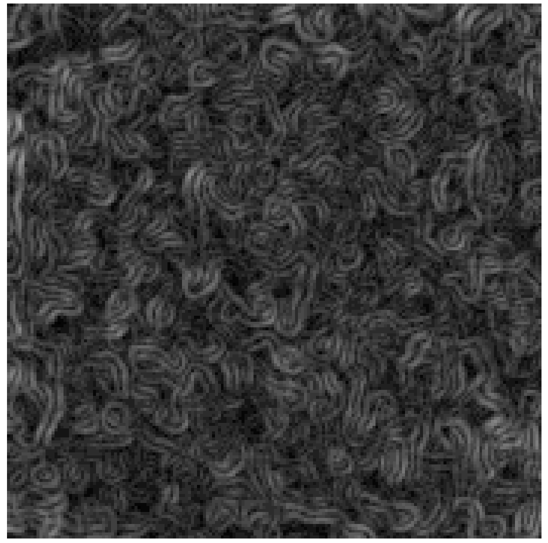
D: Mipmapping selects the suitable image resolution based on the average z value of a triangle.

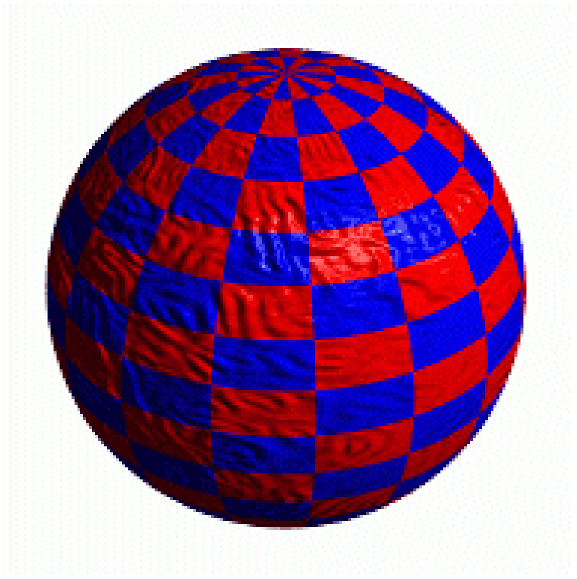
E: Alpha mapping select which pixels to render in the fragment shader.

Modulating Normals: Bump Mapping

- Derive normal perturbation from grayscale “height field”
- Emulate slight raising and lowering of surface points

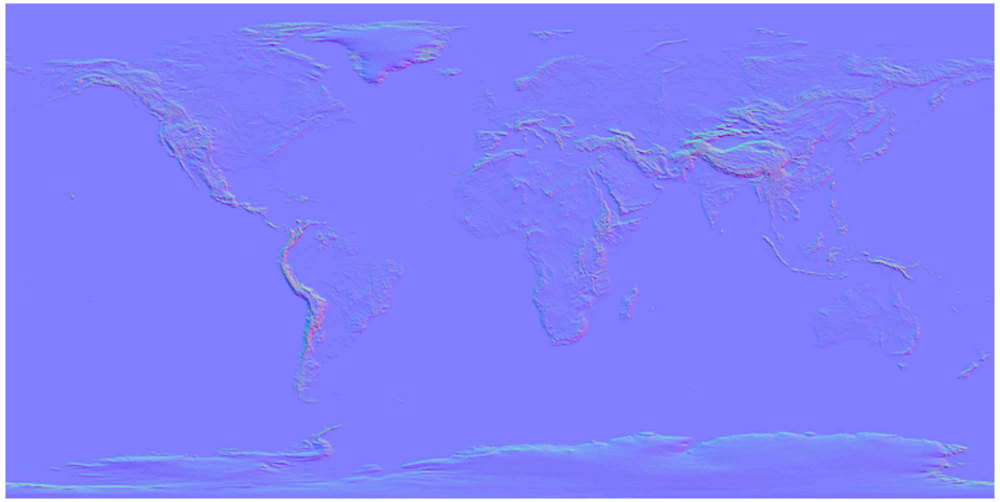

normal rendering


bump map


bump-mapped result

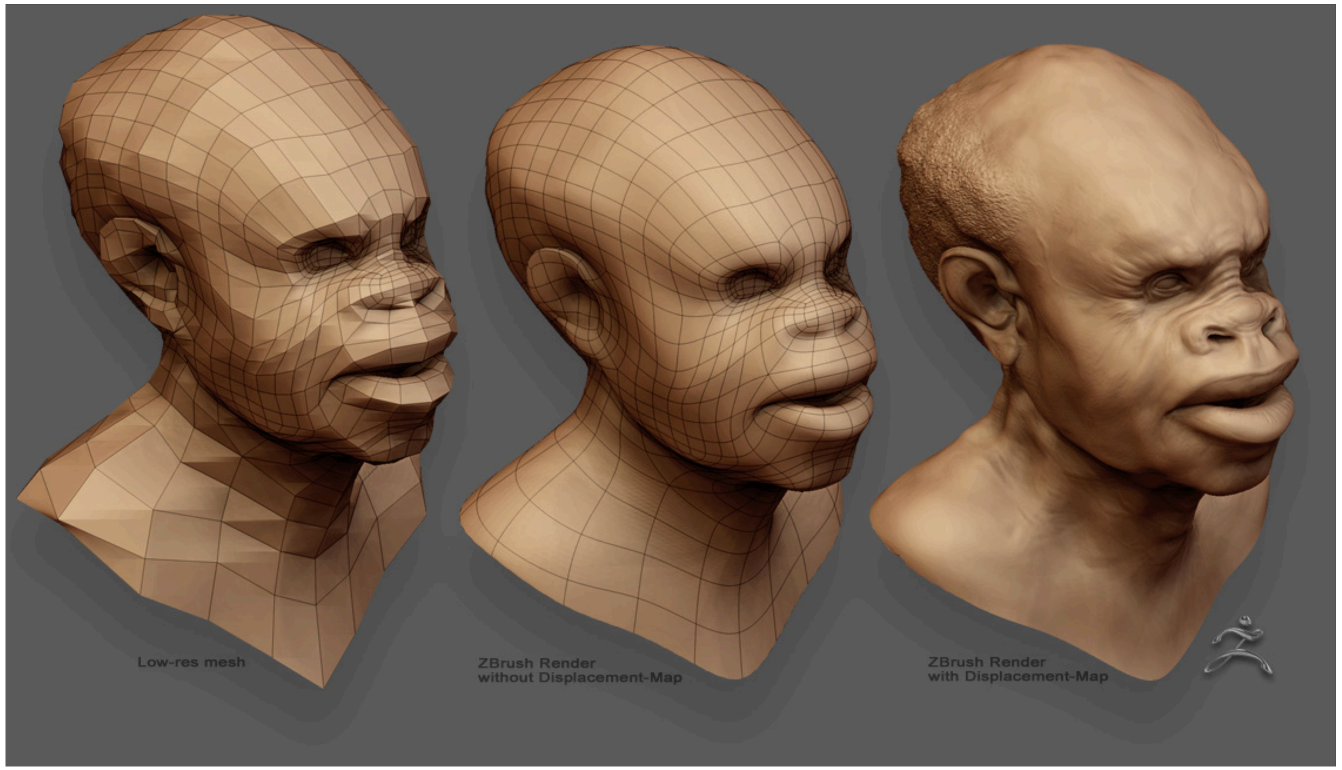
Modulating Normals: Normal Map

- Read surface normal perturbation from an RGB image



Modulating Geometry

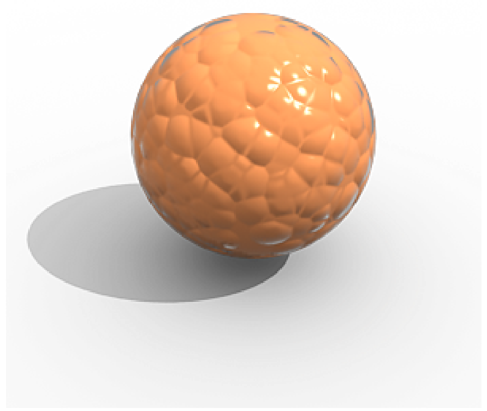
- Displacement mapping: perturb surface points
  - Need a high-resolution surface tessellation for good detail...
  - Mesh subdivision can now be done on the GPU!


Low-res mesh
ZBrush Render without Displacement-Map
ZBrush Render with Displacement-Map

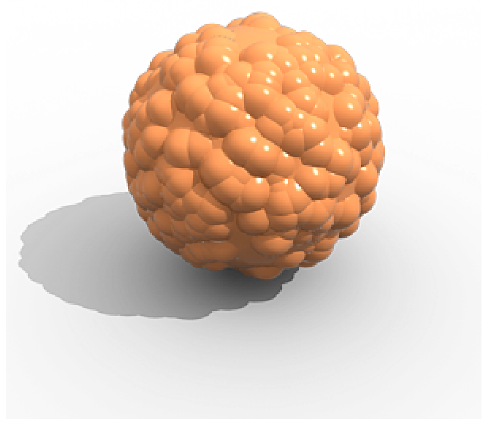
ZBrushCentral

Normal Maps vs. Displacement Maps

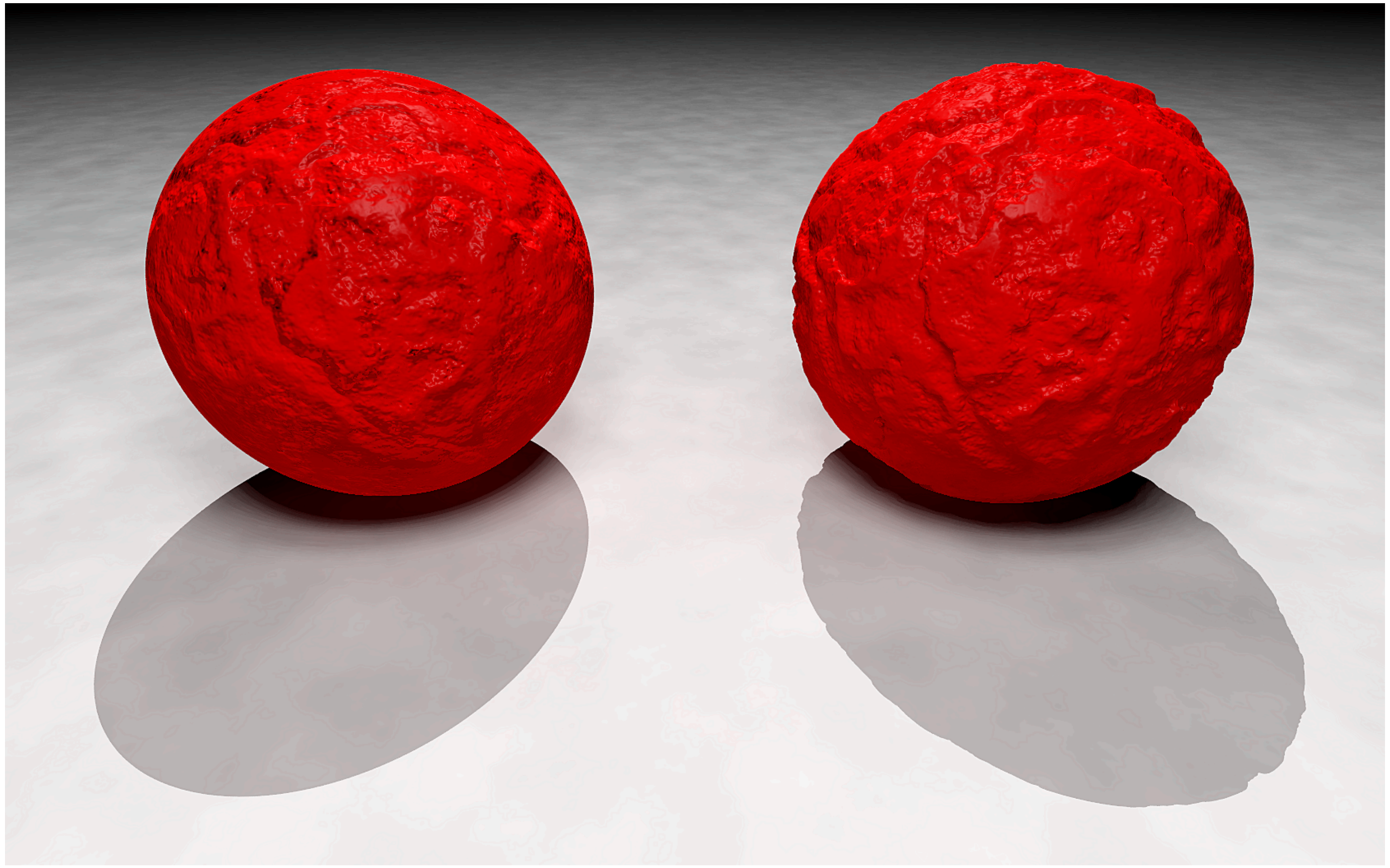
- **Normal mapping**
  - Don’t change geometry, only change normals based on texture
  - Can be performed in pixel shader
  - Silhouette still looks wrong


Normal Mapping

- **Displacement mapping**
  - Displace vertices based on offset stored in texture
  - Compute normal vectors of displaced surface
  - Performed in geometry shader or tesselation shader
  - Silhouette looks ok, but much more expensive to compute

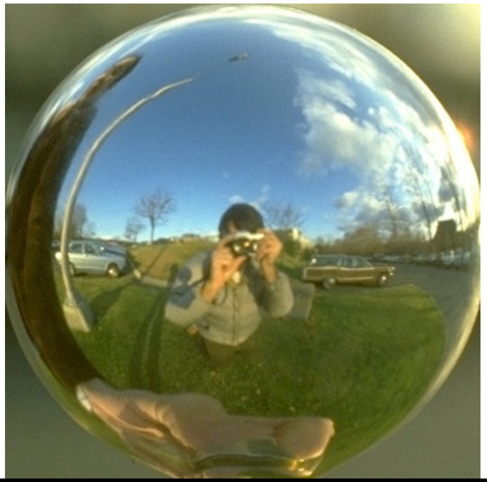

Displacement Mapping

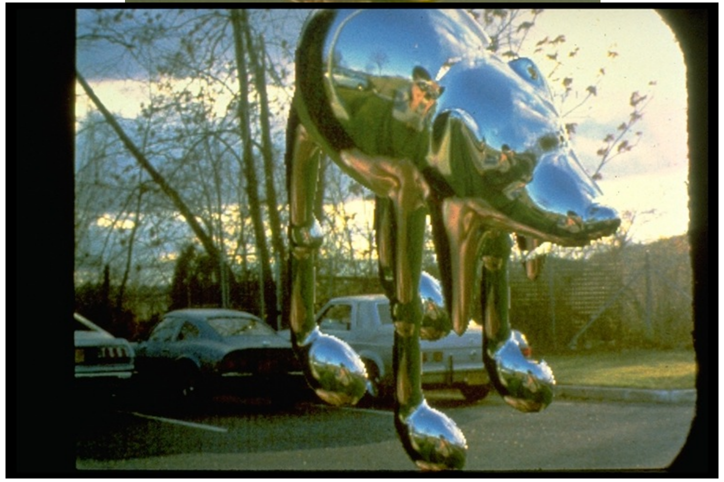
Normal Maps vs. Displacement Maps



Material Capture, Lit Sphere Rendering

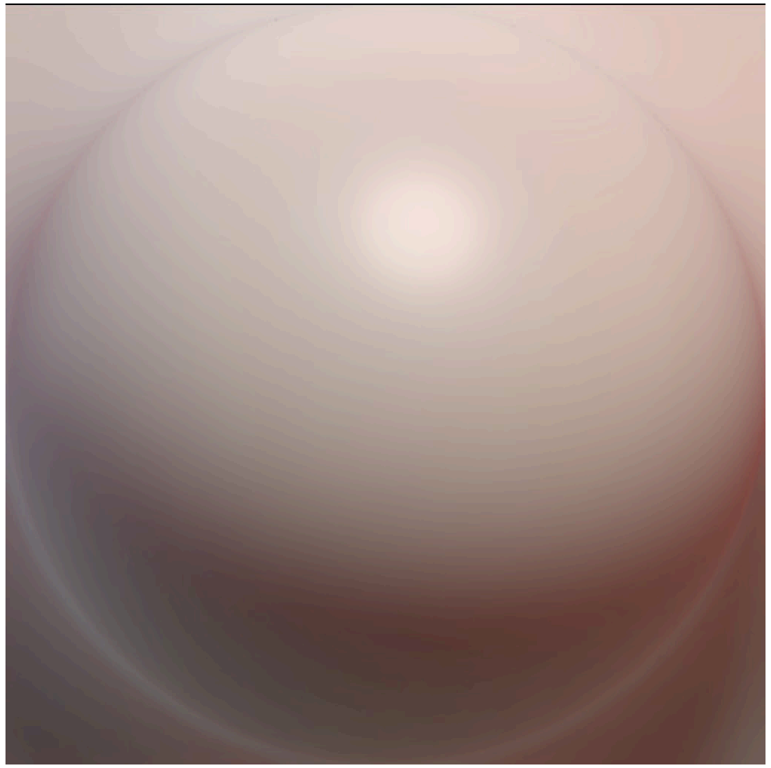
- Capture complex light-material interaction in a pre-computed texture
  - Photograph or render sphere with desired material
  - Use this image as spherical environment map
  - Access texture through normal vector
- Good: Allows complex material with simple (therefore efficient) shader
- Bad: Assume viewer is far away, does not handle surface micro-structure





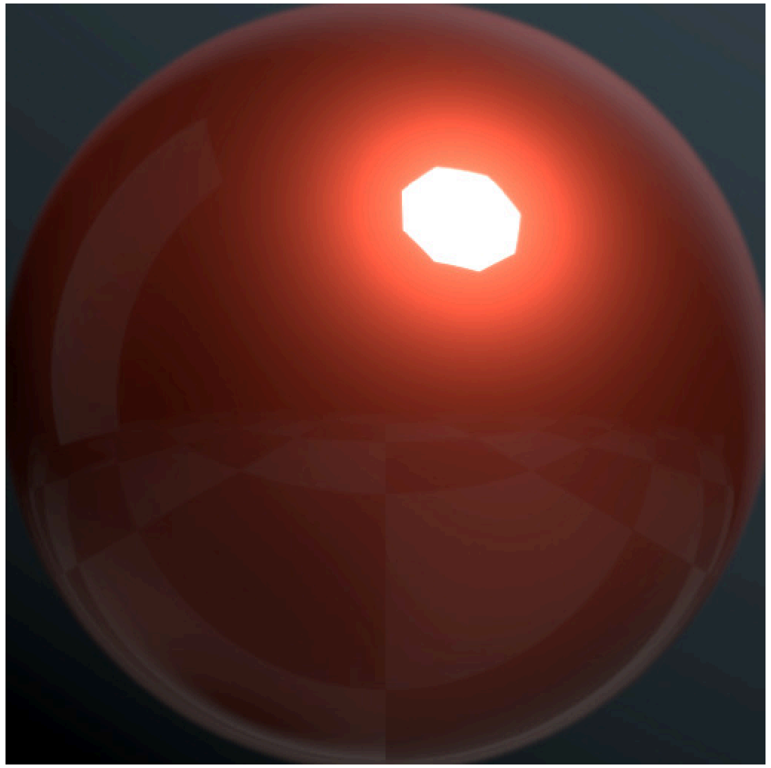
Sloan et al., The Lit Sphere, Graphics Interface 2001

Material Capture, Lit Sphere Rendering


skin

MatCap from Blender 2.81 repository

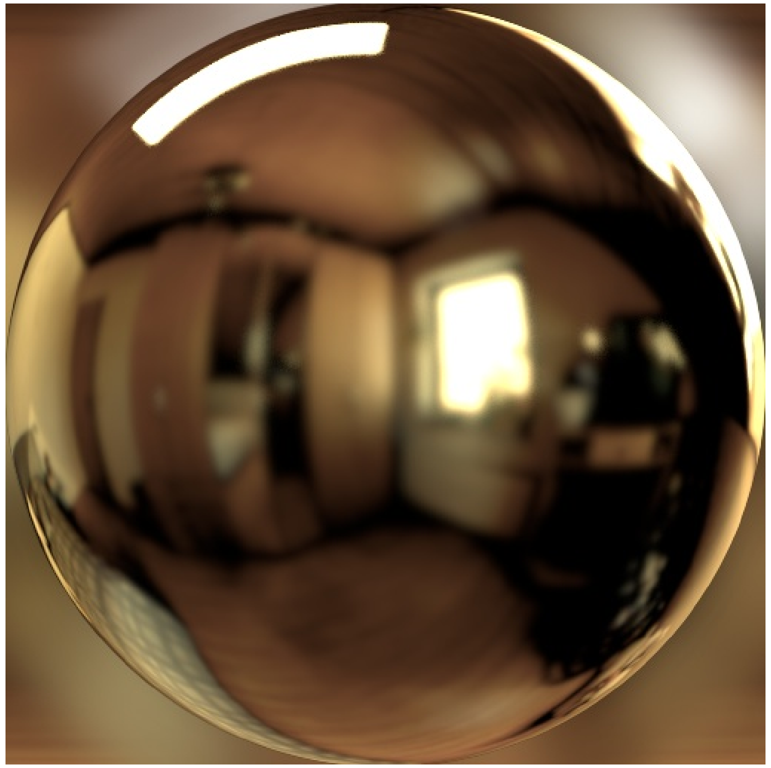
Material Capture, Lit Sphere Rendering



car paint

MatCap from Blender 2.81 repository

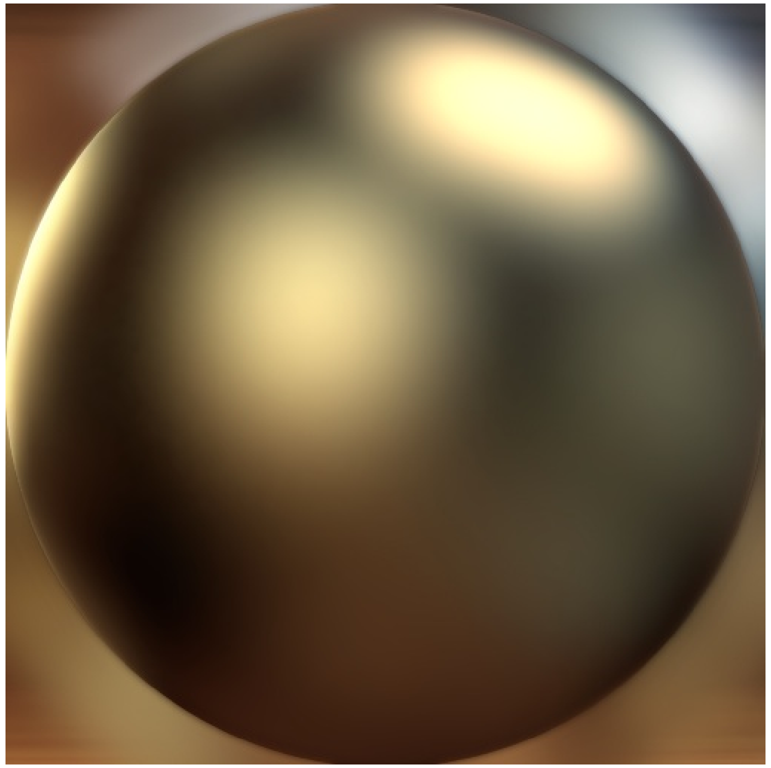
Material Capture, Lit Sphere Rendering



shiny gold

MatCap from Blender 2.79 repository

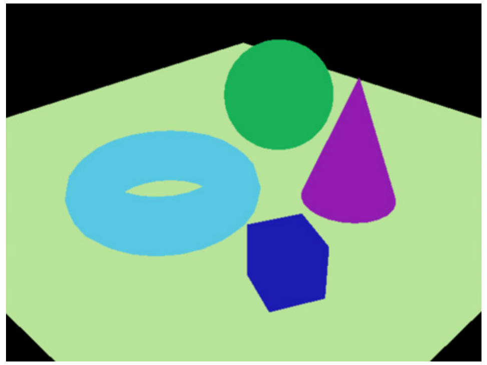
Material Capture, Lit Sphere Rendering

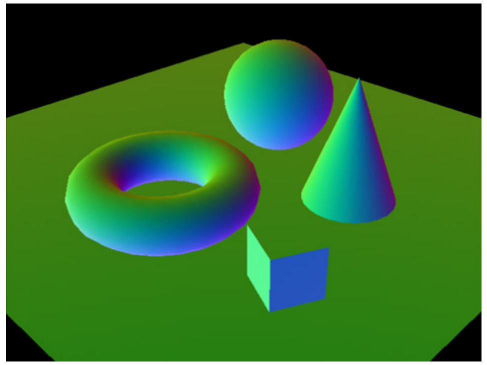

matte gold

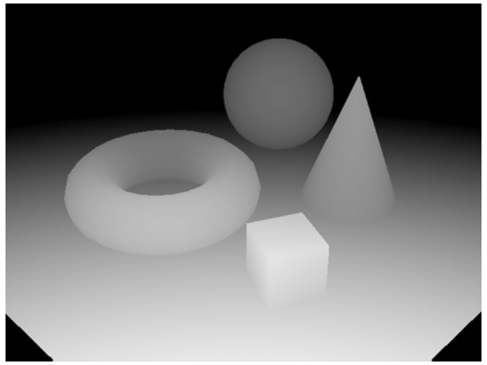
MatCap from Blender 2.79 repository

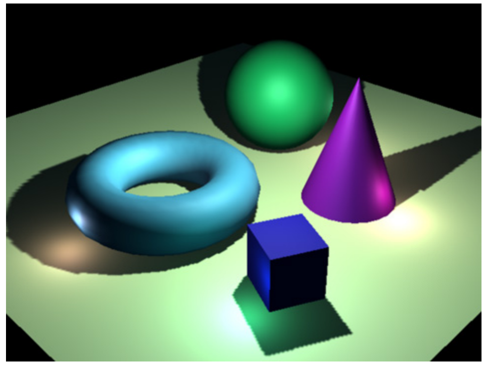
Deferred Shading

- Screen-space lighting/shading technique
  1. Render colors, normals, and depth into textures
  2. Use screen-space filter to compute lighting for each image pixel
- Good: only compute lighting for visible pixels
- Bad: transparency and antialiasing are more difficult

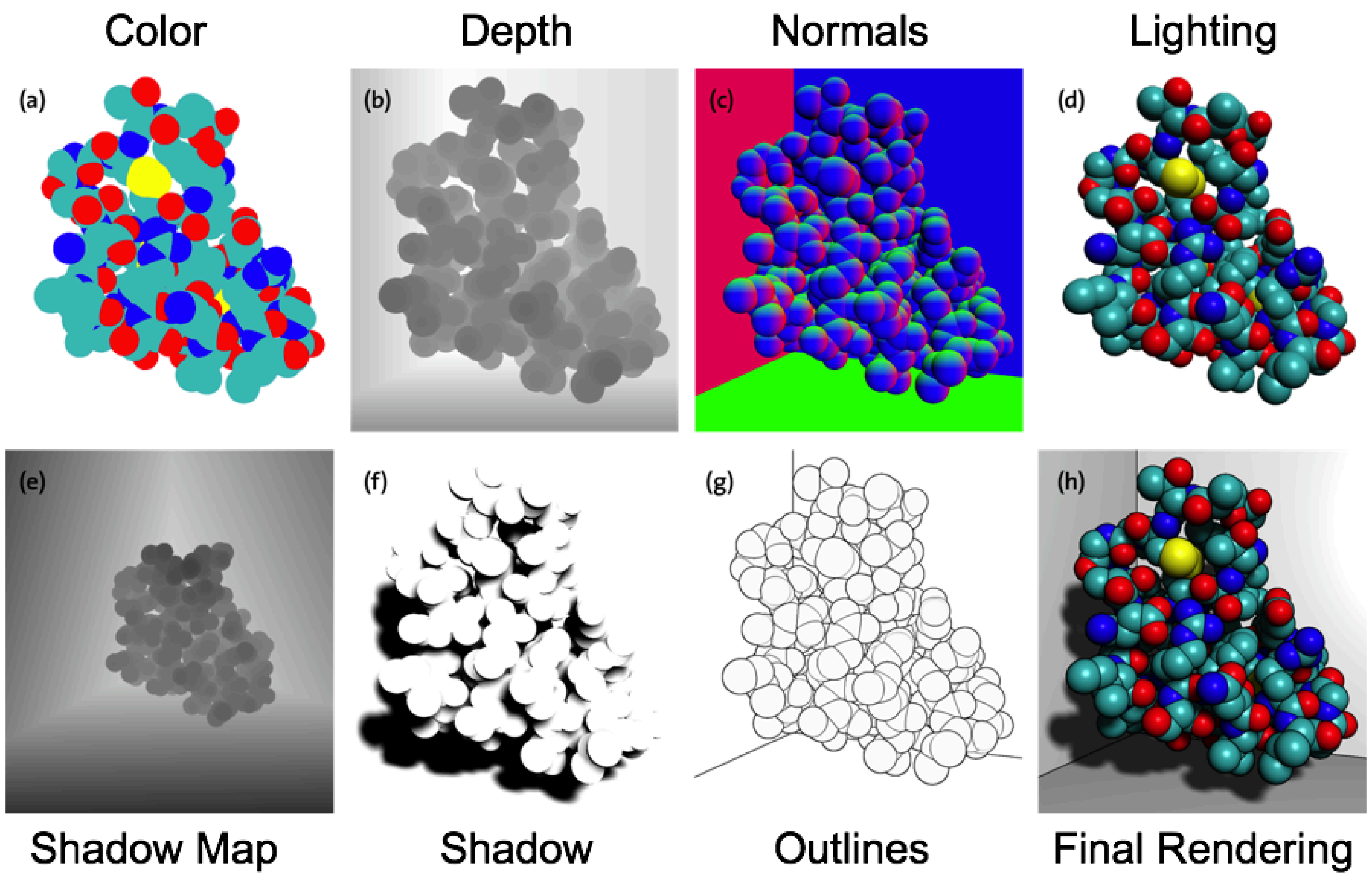

Colors


Normals


Depth

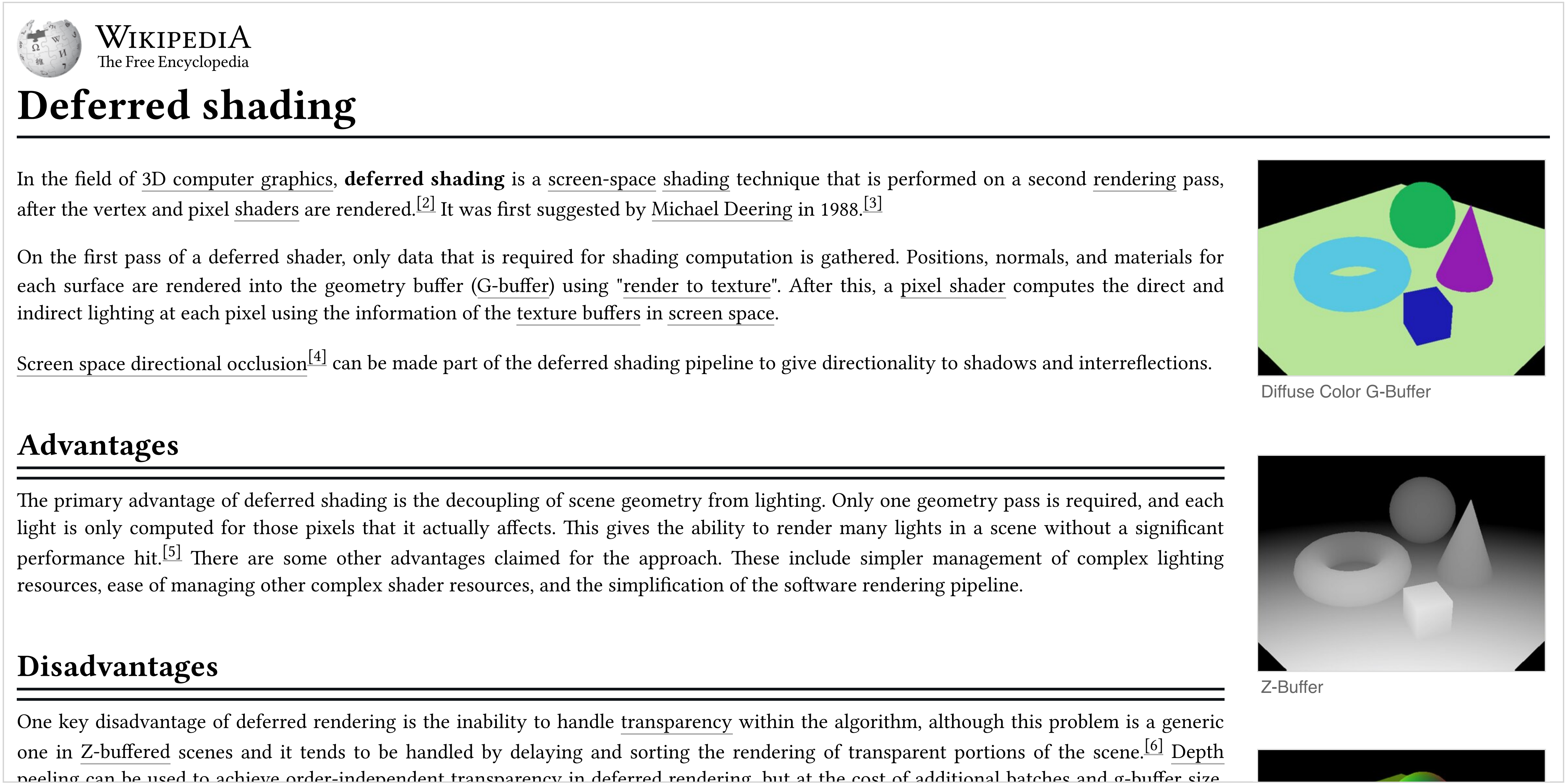

Final result

Deferred Shading



Sigg et al, “GPU-Based Ray-Casting of Quadratic Surfaces”, PBG 2006

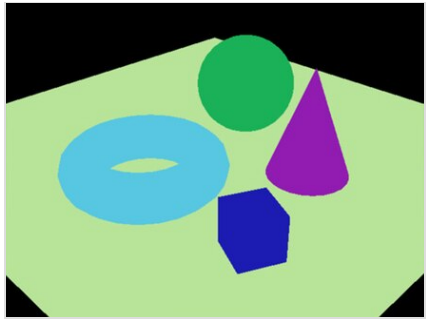
Deferred Shading



Wikipedia
The Free Encyclopedia

Deferred shading

In the field of 3D computer graphics, **deferred shading** is a screen-space shading technique that is performed on a second rendering pass, after the vertex and pixel shaders are rendered.[2] It was first suggested by Michael Deering in 1988.[3]

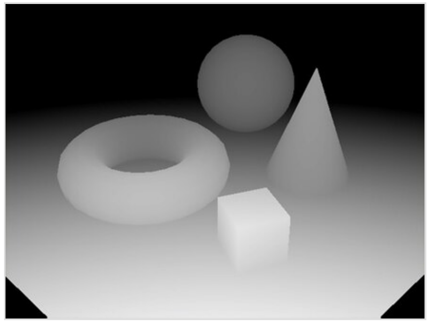

Diffuse Color G-Buffer

On the first pass of a deferred shader, only data that is required for shading computation is gathered. Positions, normals, and materials for each surface are rendered into the geometry buffer (G-buffer) using "render to texture". After this, a pixel shader computes the direct and indirect lighting at each pixel using the information of the texture buffers in screen space.

Screen space directional occlusion[4] can be made part of the deferred shading pipeline to give directionality to shadows and interreflections.

Advantages

The primary advantage of deferred shading is the decoupling of scene geometry from lighting. Only one geometry pass is required, and each light is only computed for those pixels that it actually affects. This gives the ability to render many lights in a scene without a significant performance hit.[5] There are some other advantages claimed for the approach. These include simpler management of complex lighting resources, ease of managing other complex shader resources, and the simplification of the software rendering pipeline.


Z-Buffer

Disadvantages

One key disadvantage of deferred rendering is the inability to handle transparency within the algorithm, although this problem is a generic one in Z-buffered scenes and it tends to be handled by delaying and sorting the rendering of transparent portions of the scene.[6] Depth peeling can be used to achieve order-independent transparency in deferred rendering, but at the cost of additional batches and g-buffer size.

Source

Deferred Shading

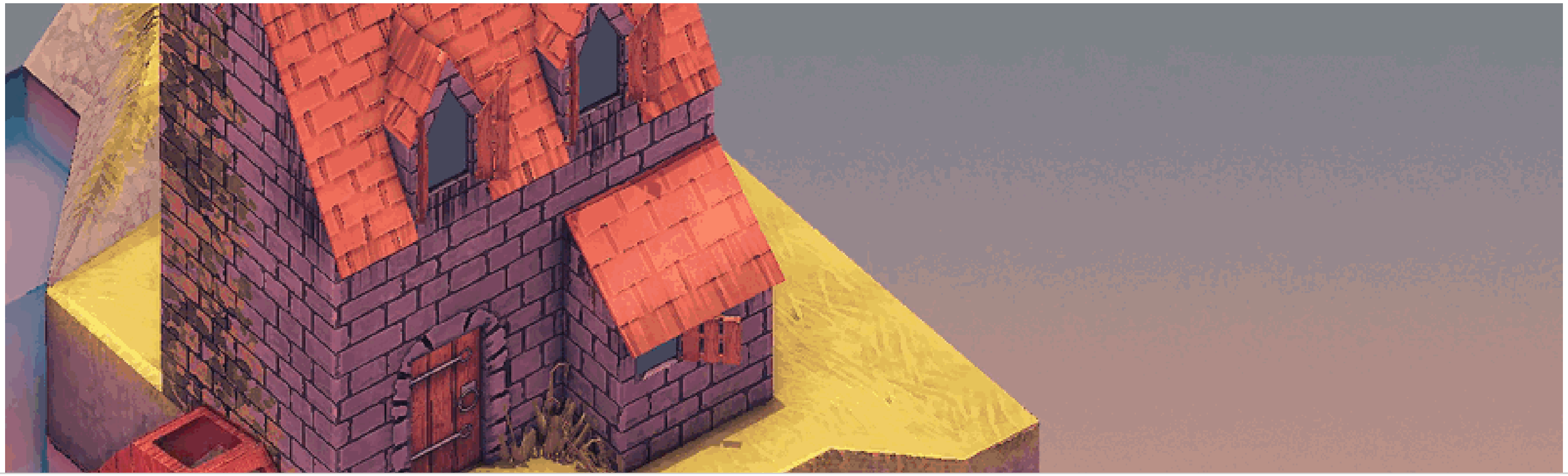
Source

Screen Space Reflections

- Simplified reflections computed using the depth buffer.

3D Game Shaders For Beginners

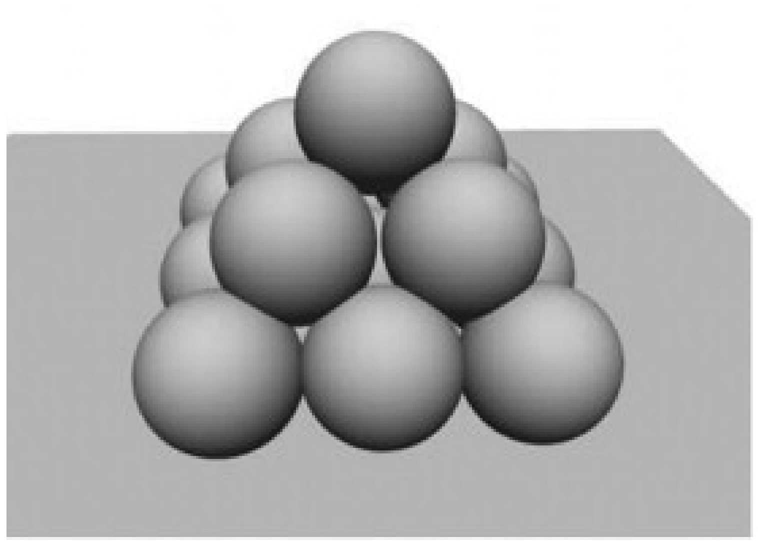
Screen Space Reflection (SSR)

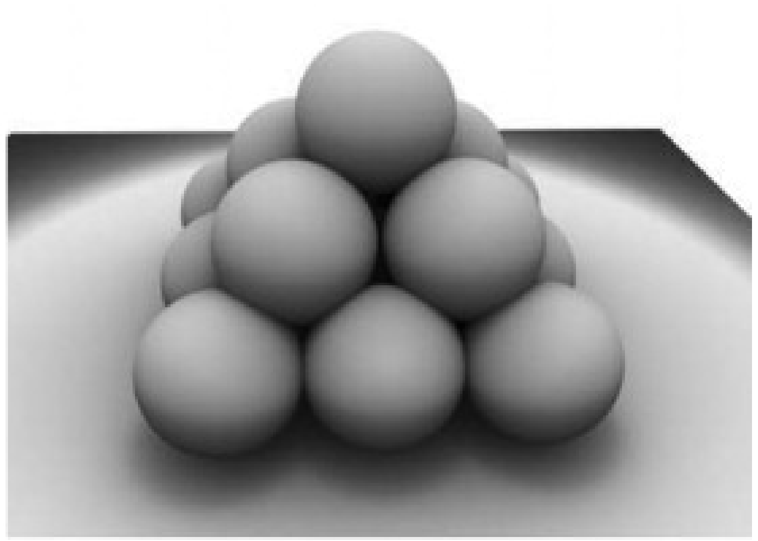


Source

Ambient Occlusion

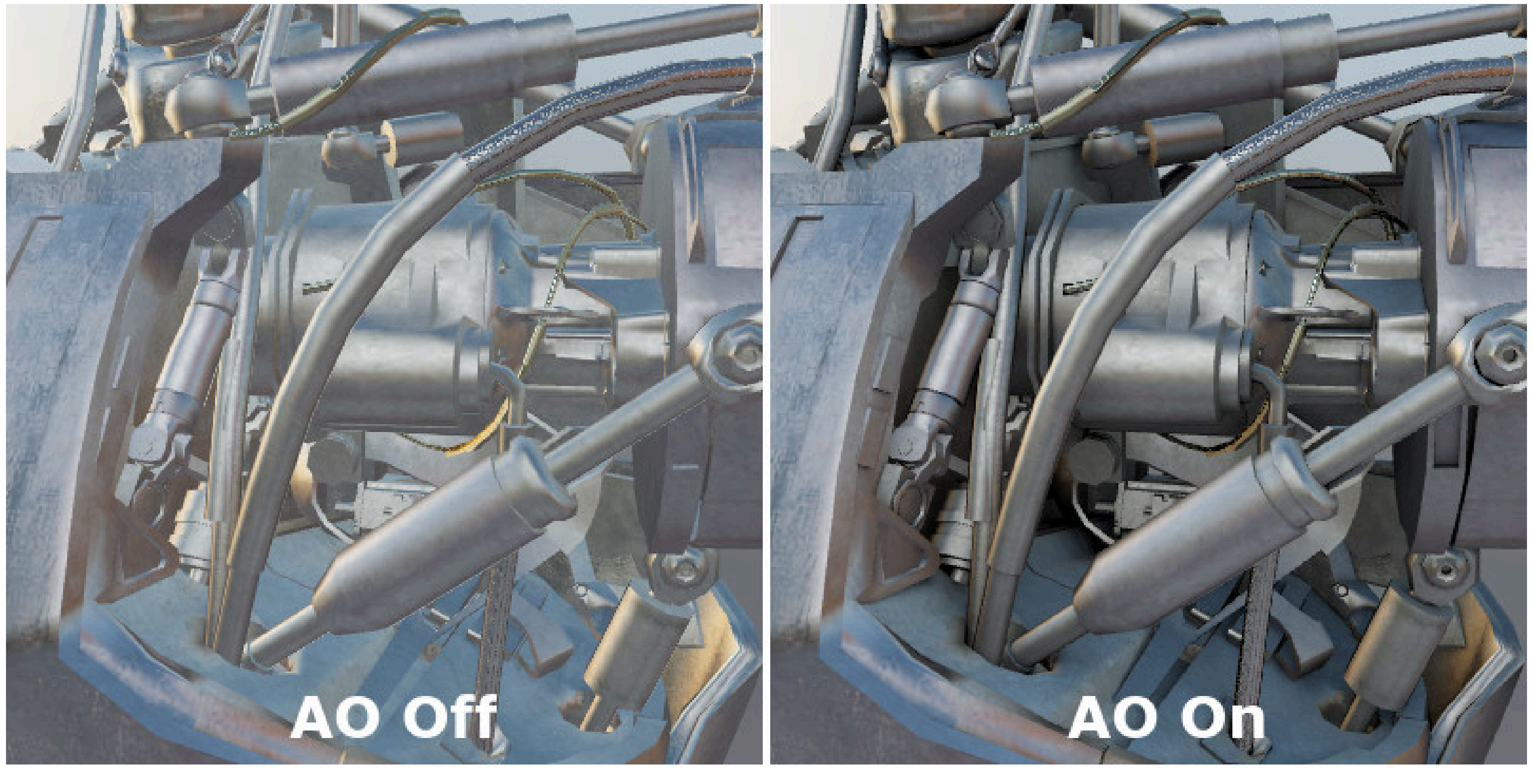
- Measures the exposure to ambient lighting to darken more occluded, less acessible parts of the scene. Global method, can be efficiently implemented in screen space.


Without ambient occlusion


With ambient occlusion

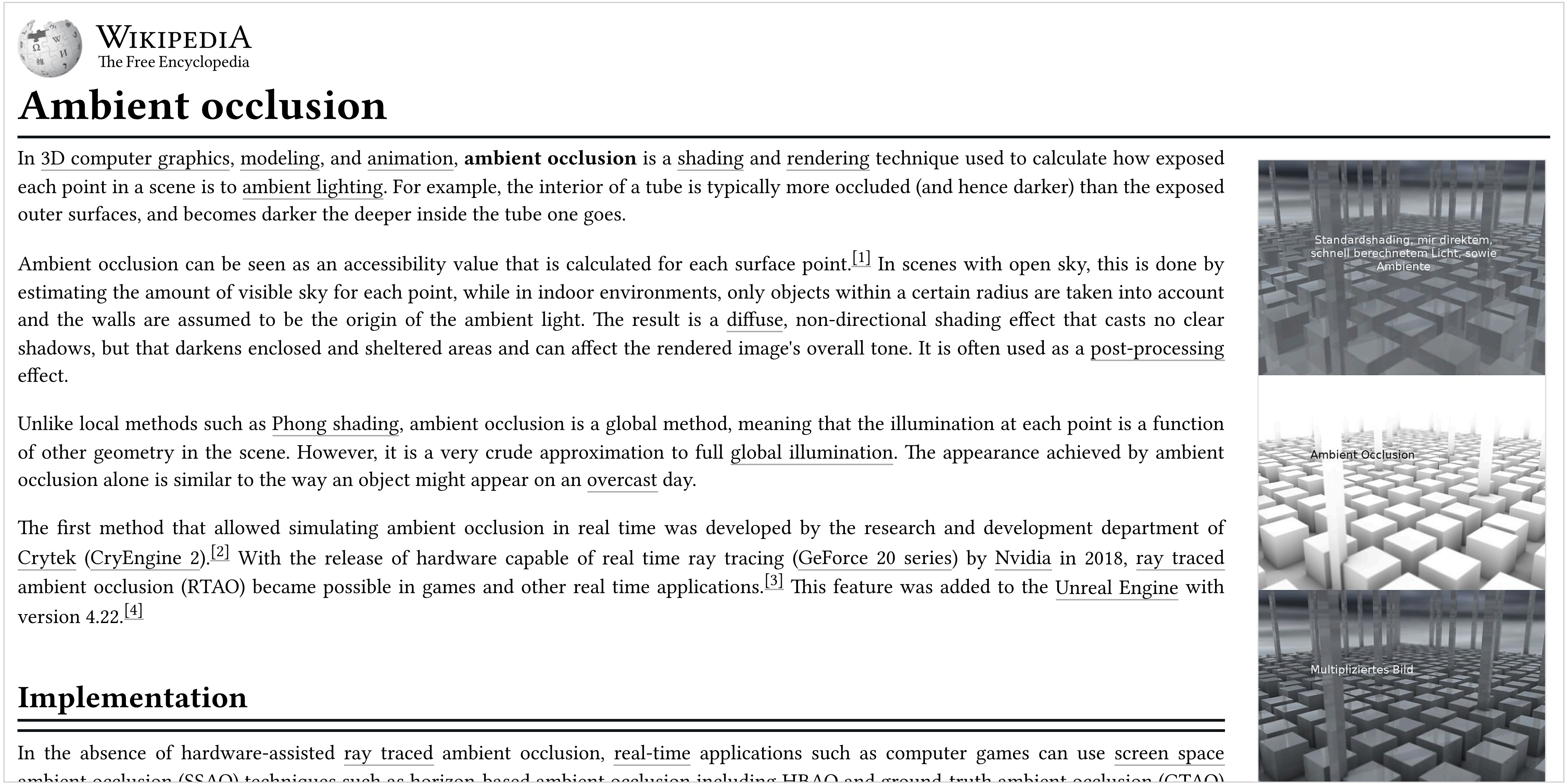
Ambient Occlusion

- Measures the exposure to ambient lighting to darken more occluded, less acessible parts of the scene. Global method, can be efficiently implemented in screen space.



Source

Ambient Occlusion



WIKIPEDIA
The Free Encyclopedia

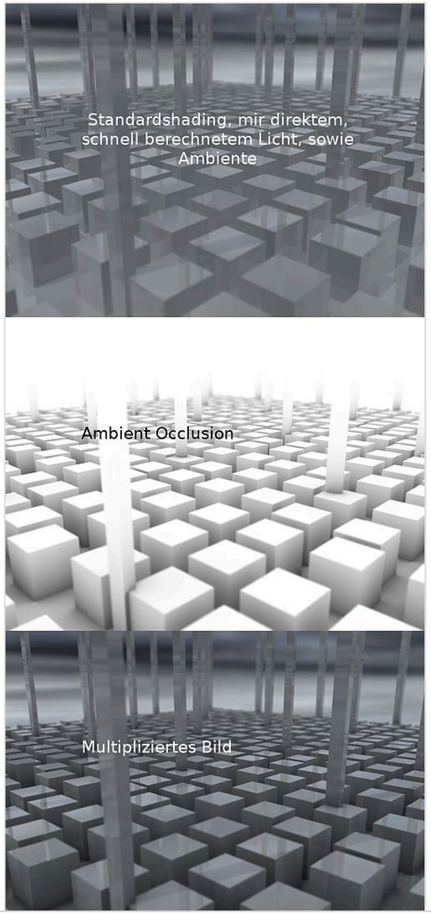
Ambient occlusion

In 3D computer graphics, modeling, and animation, **ambient occlusion** is a shading and rendering technique used to calculate how exposed each point in a scene is to ambient lighting. For example, the interior of a tube is typically more occluded (and hence darker) than the exposed outer surfaces, and becomes darker the deeper inside the tube one goes.

Ambient occlusion can be seen as an accessibility value that is calculated for each surface point.[1] In scenes with open sky, this is done by estimating the amount of visible sky for each point, while in indoor environments, only objects within a certain radius are taken into account and the walls are assumed to be the origin of the ambient light. The result is a diffuse, non-directional shading effect that casts no clear shadows, but that darkens enclosed and sheltered areas and can affect the rendered image's overall tone. It is often used as a post-processing effect.

Unlike local methods such as Phong shading, ambient occlusion is a global method, meaning that the illumination at each point is a function of other geometry in the scene. However, it is a very crude approximation to full global illumination. The appearance achieved by ambient occlusion alone is similar to the way an object might appear on an overcast day.

The first method that allowed simulating ambient occlusion in real time was developed by the research and development department of Crytek (CryEngine 2).[2] With the release of hardware capable of real time ray tracing (GeForce 20 series) by Nvidia in 2018, ray traced ambient occlusion (RTAO) became possible in games and other real time applications.[3] This feature was added to the Unreal Engine with version 4.22.[4]


Standardshading, mir direktem, schnell berechnetem Licht, sowie Ambiente
Ambient Occlusion
Multipliziertes Bild

Implementation

In the absence of hardware-assisted ray traced ambient occlusion, real-time applications such as computer games can use screen space ambient occlusion (SSAO) techniques such as horizon-based ambient occlusion including HBAO and ground-truth ambient occlusion (GTAO)

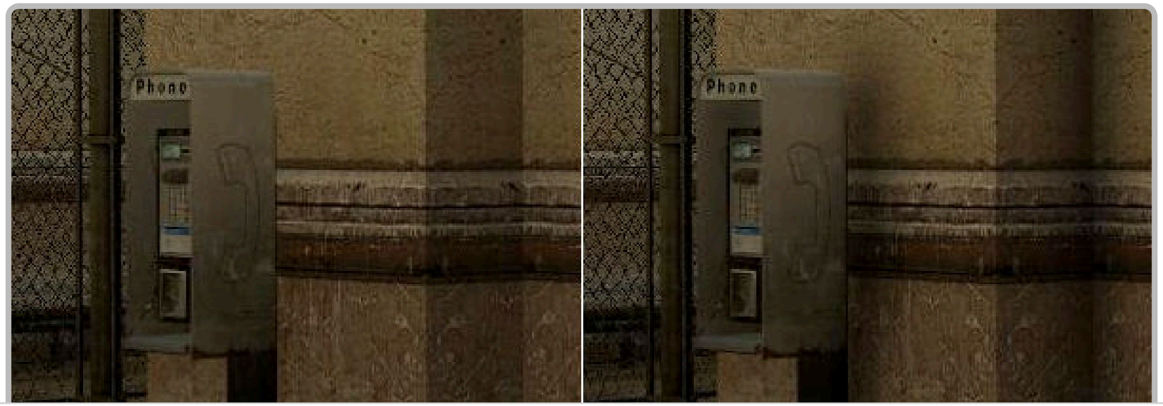
Wikipedia

Ambient Occlusion

SSAO

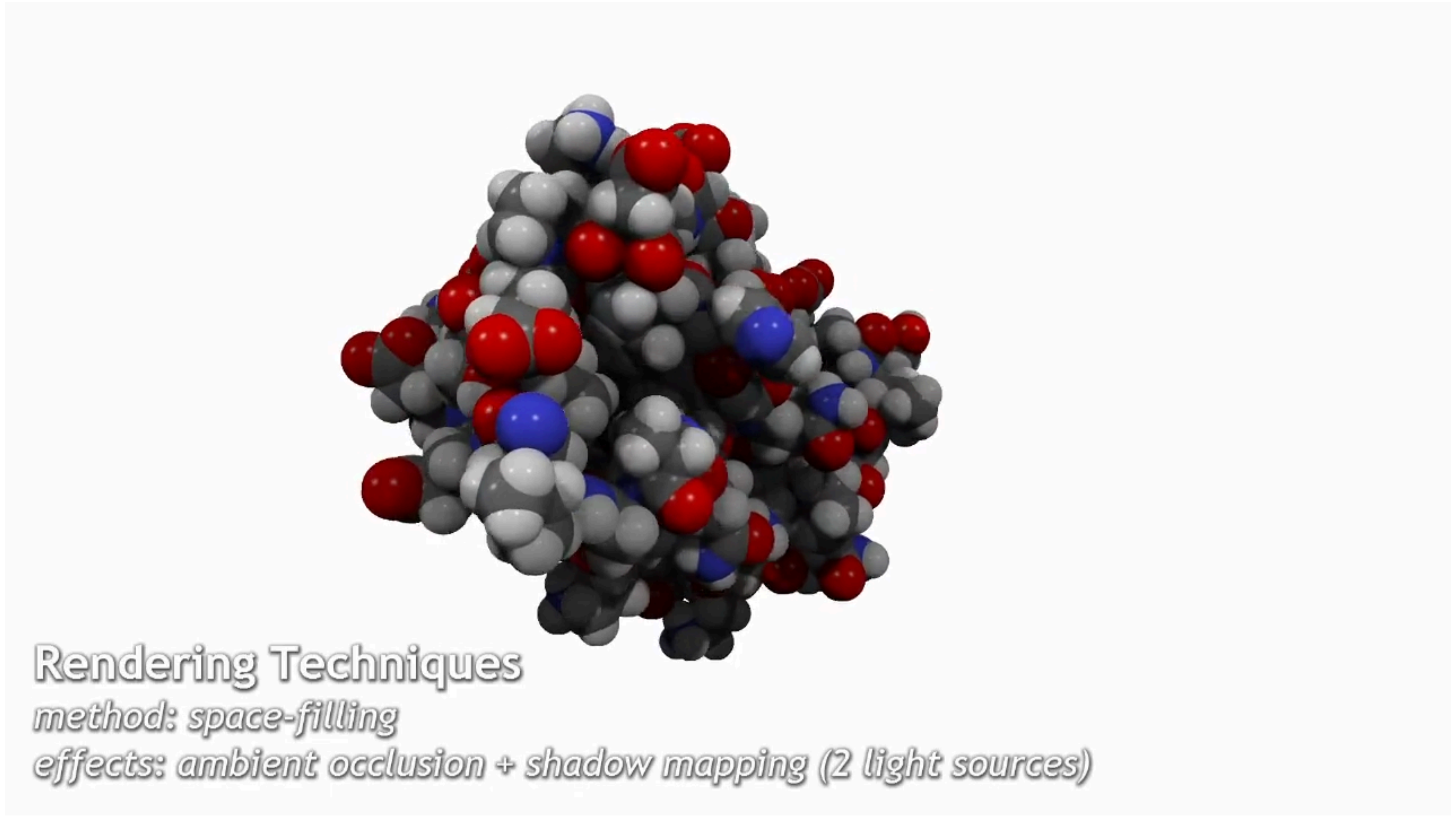
We've briefly touched the topic in the basic lighting chapter: ambient lighting. Ambient lighting is a fixed light constant we add to the overall lighting of a scene to simulate the scattering of light. In reality, light scatters in all kinds of directions with varying intensities so the indirectly lit parts of a scene should also have varying intensities. One type of indirect lighting approximation is called ambient occlusion that tries to approximate indirect lighting by darkening creases, holes, and surfaces that are close to each other. These areas are largely occluded by surrounding geometry and thus light rays have fewer places to escape to, hence the areas appear darker. Take a look at the corners and creases of your room to see that the light there seems just a little darker.

Below is an example image of a scene with and without ambient occlusion. Notice how especially between the creases, the (ambient) light is more occluded:



Source

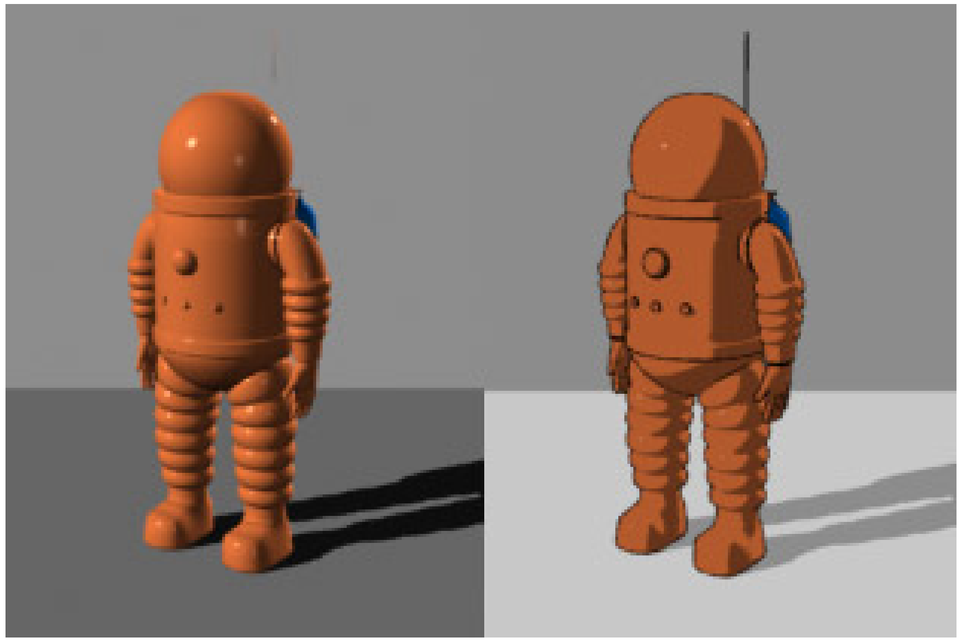
Molecule Rendering

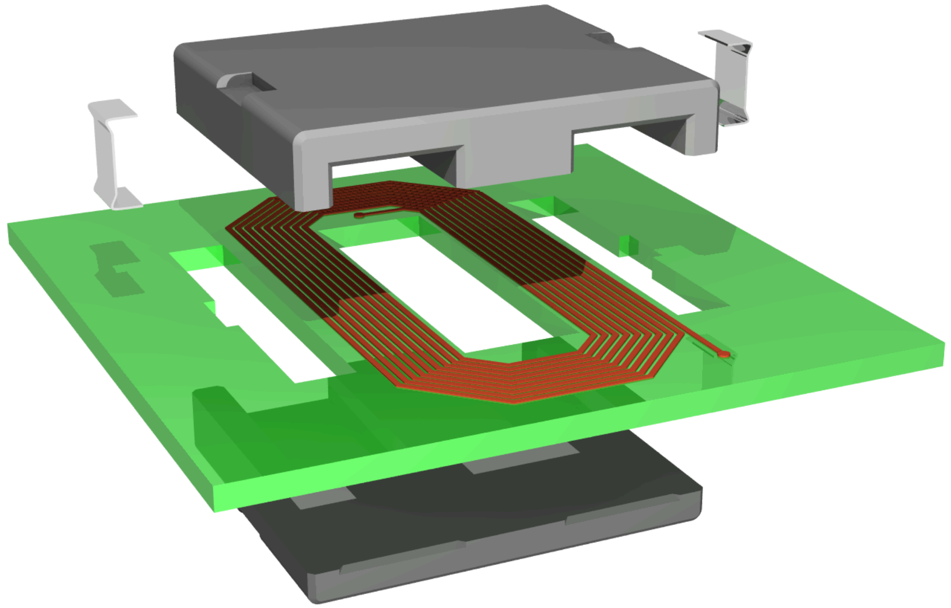

Rendering Techniques
method: space-filling
effects: ambient occlusion + shadow mapping (2 light sources)

Waltemate et al, "Membrane Mapping", VCBM 2014

Non-Photorealistic Rendering

- Often called *expressive graphics*, *artistic rendering*, *art-based rendering*, etc.

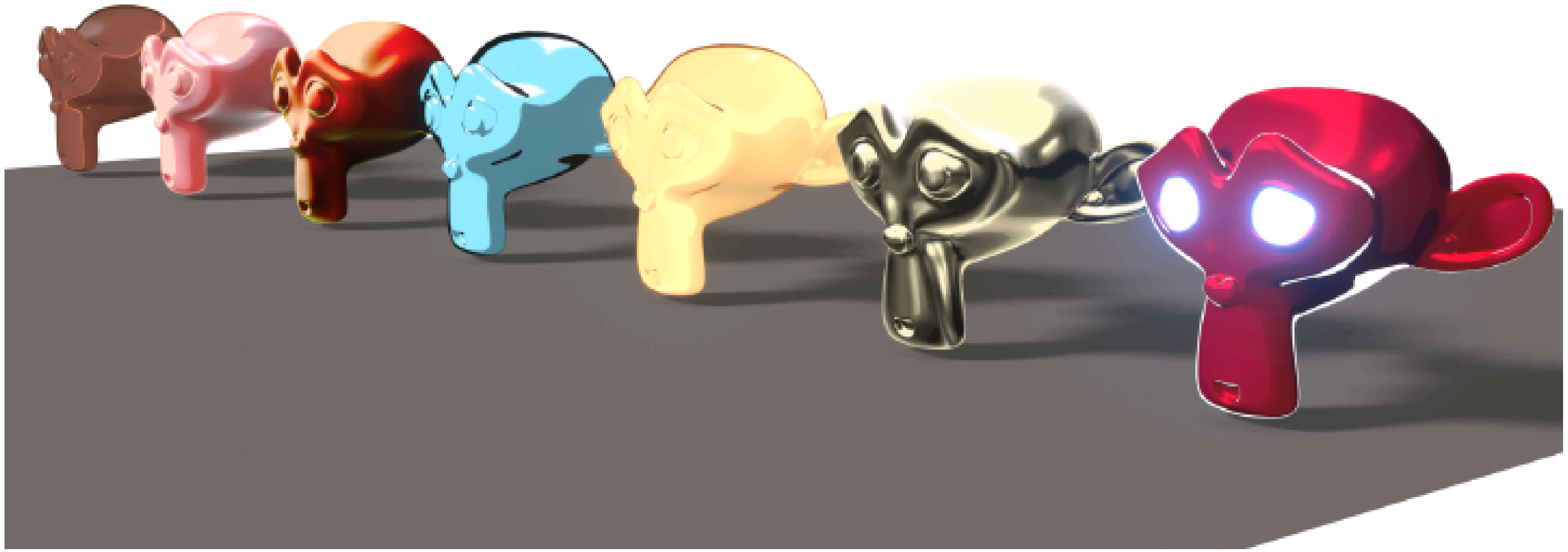

Toon Shader


Technical Illustration

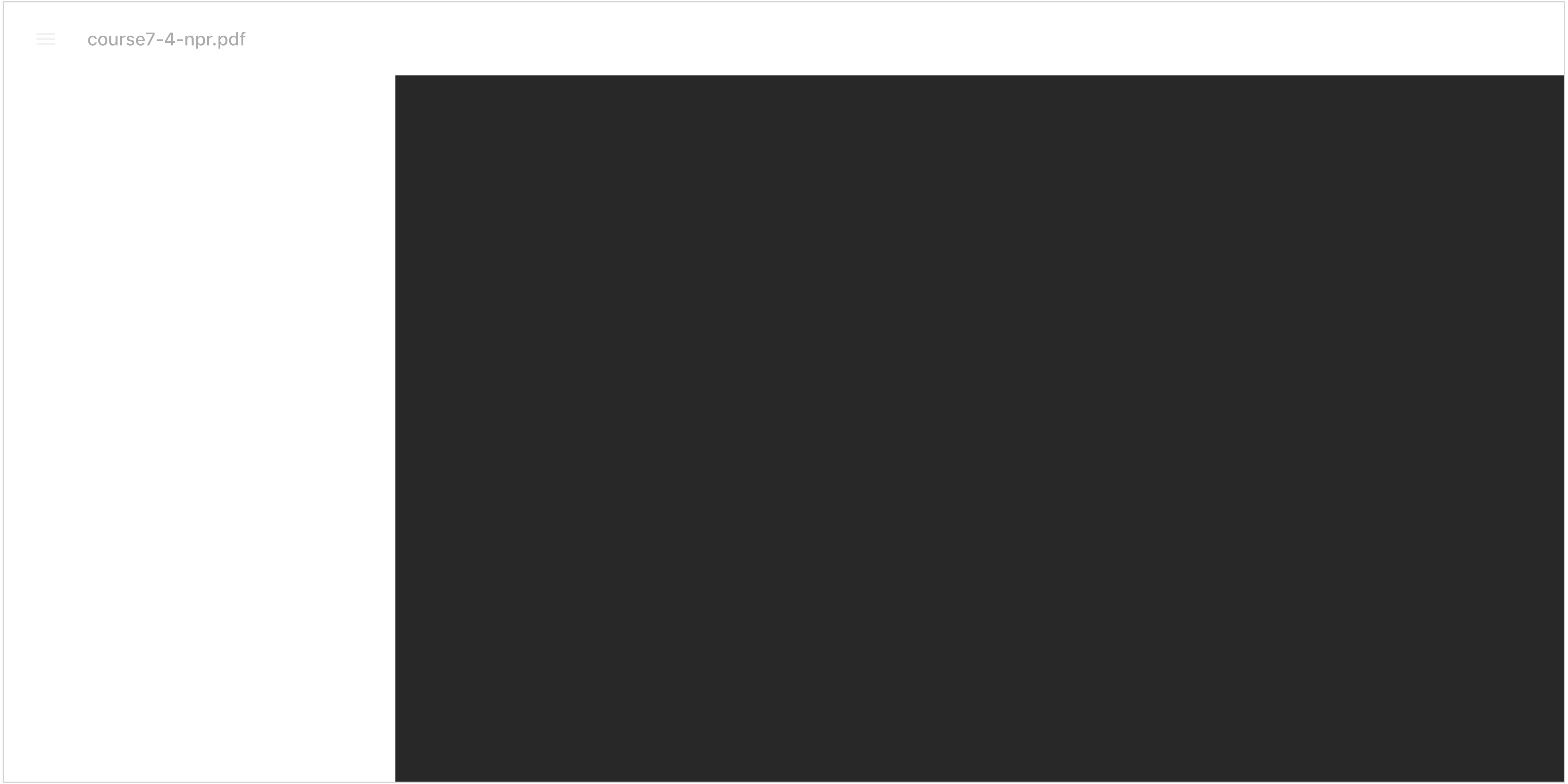
Wikipedia

Non-Photorealistic Rendering

- Often called *expressive graphics*, *artistic rendering*, *art-based rendering*, etc.

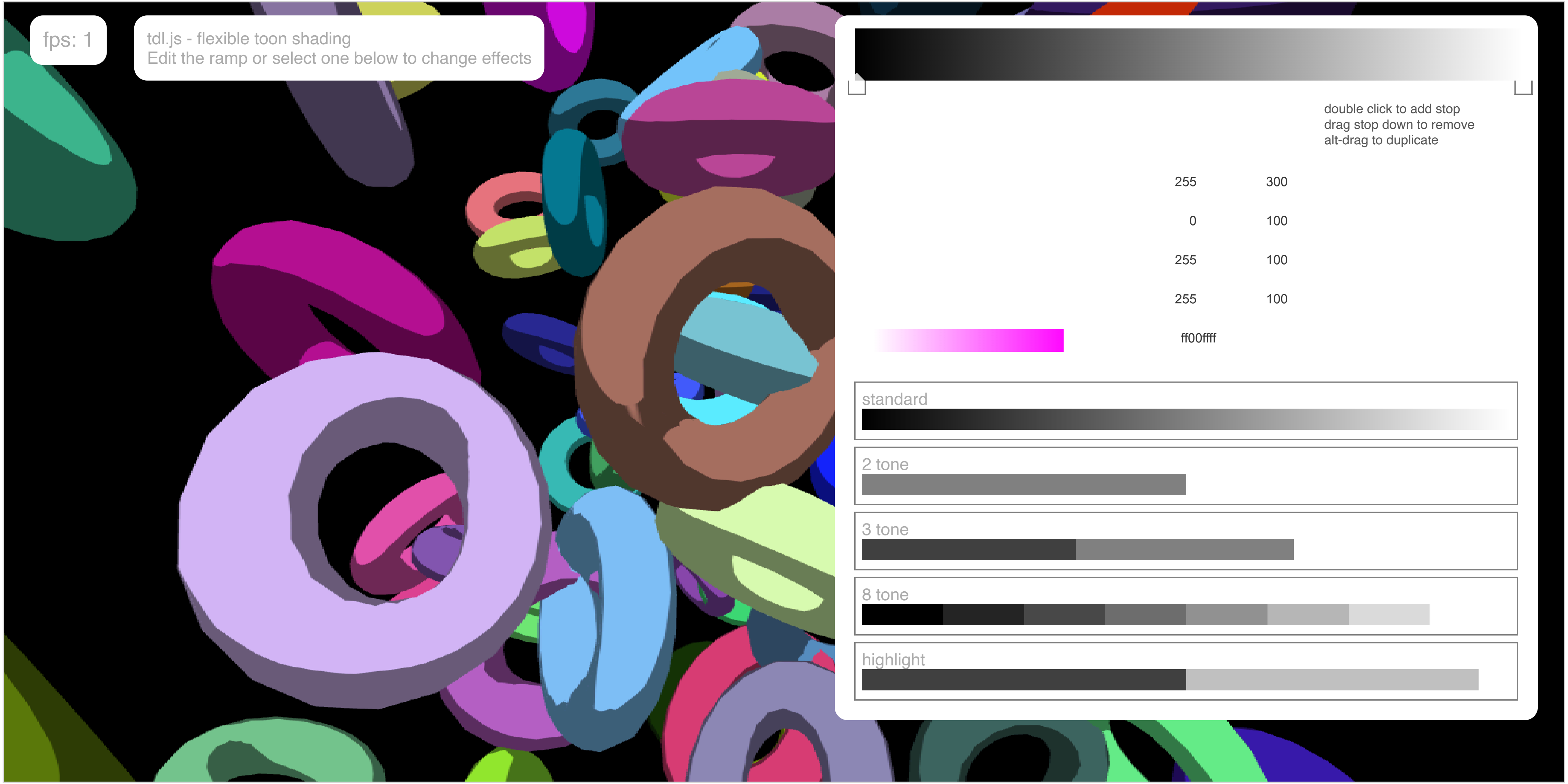

NPR in Blender

Non-Photorealistic Rendering



Source

Non-Photorealistic Rendering



fps: 1

tdl.js - flexible toon shading
Edit the ramp or select one below to change effects

double click to add stop
drag stop down to remove
alt-drag to duplicate

255	300
0	100
255	100
255	100

ff00ffff

standard

2 tone

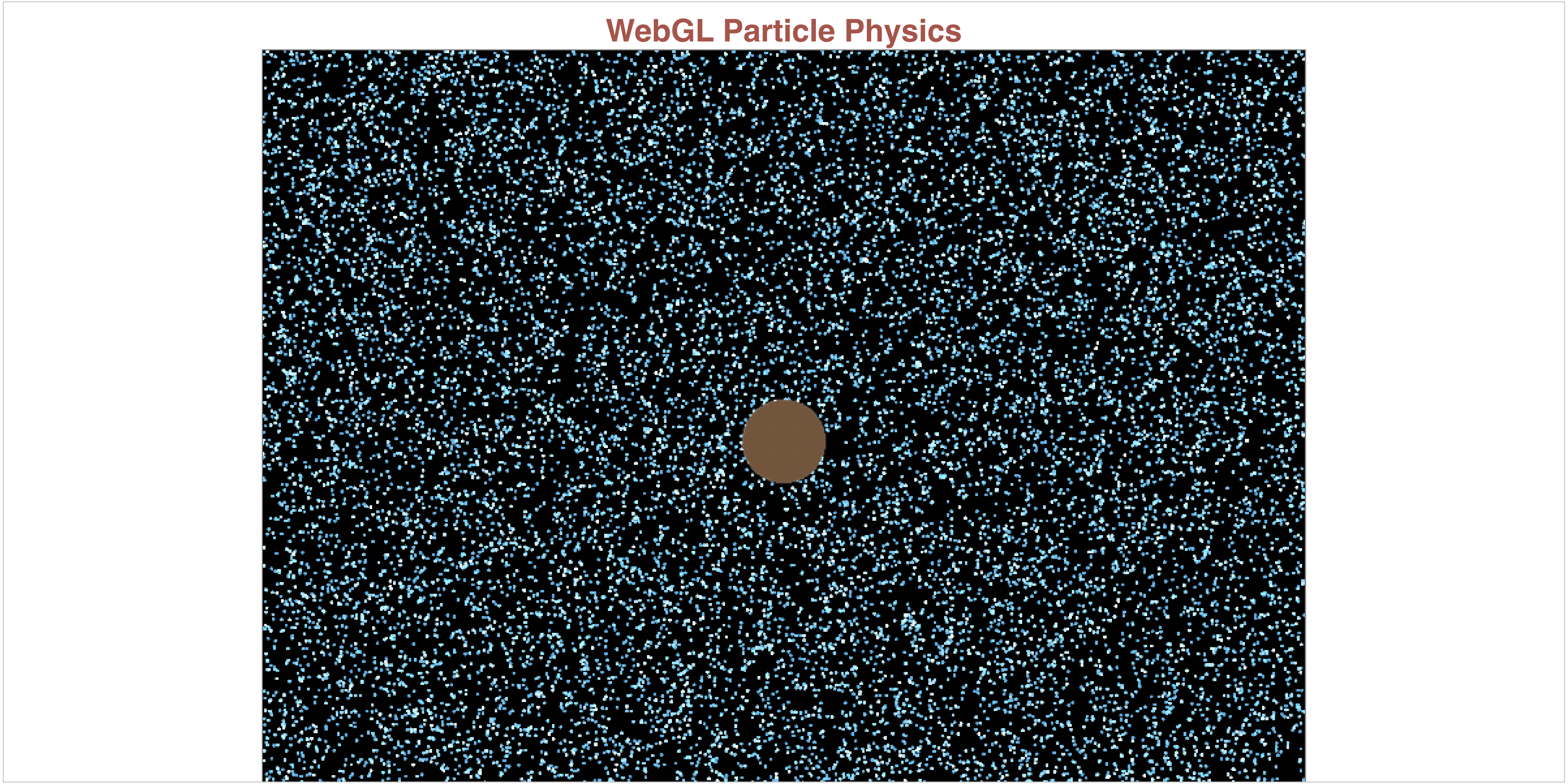
3 tone

8 tone

highlight

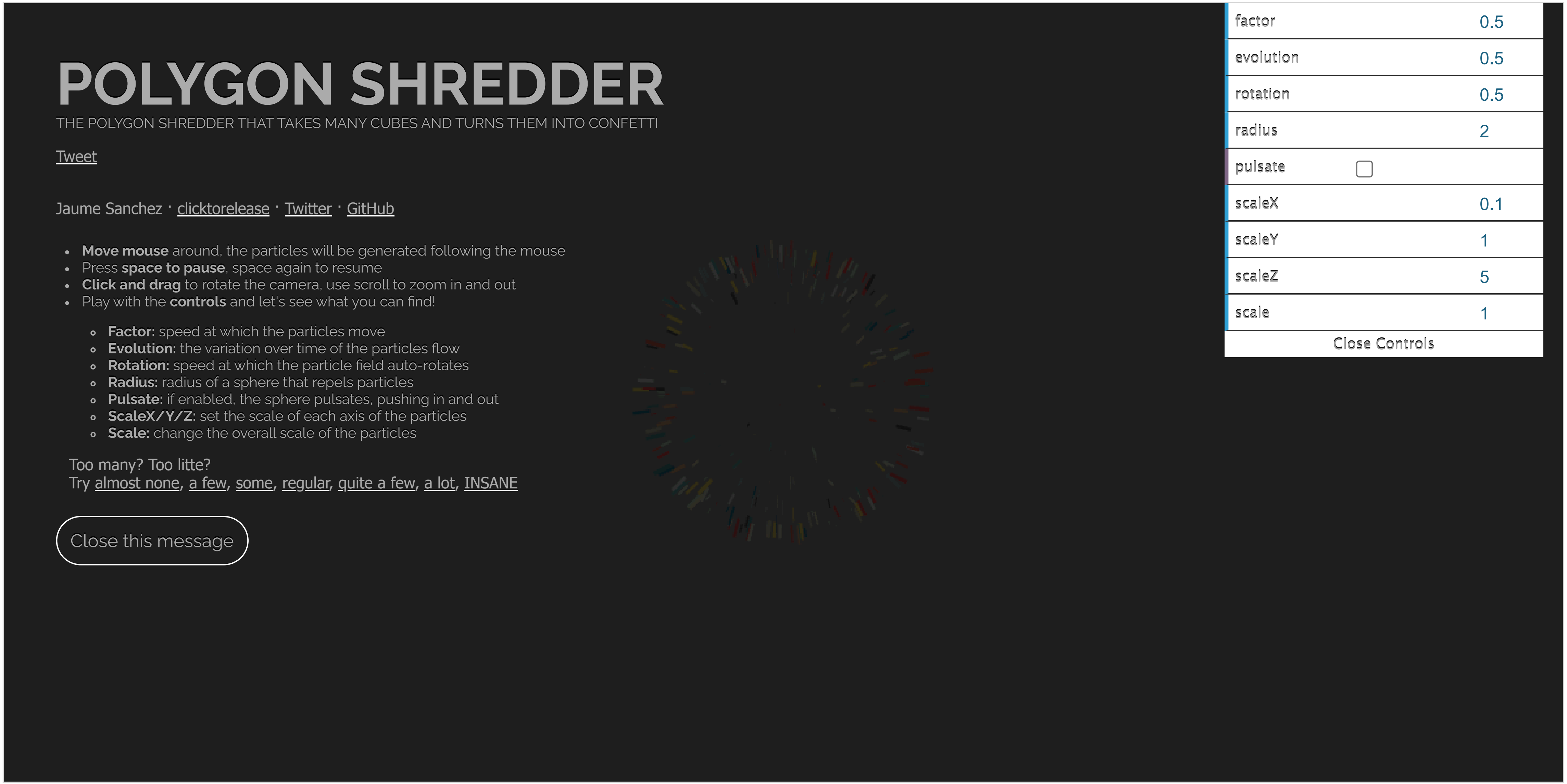
Source

Particle Systems



Source

Particle Systems



POLYGON SHREDDER

THE POLYGON SHREDDER THAT TAKES MANY CUBES AND TURNS THEM INTO CONFETTI

Tweet

Jaume Sanchez · clicktorelease · Twitter · GitHub

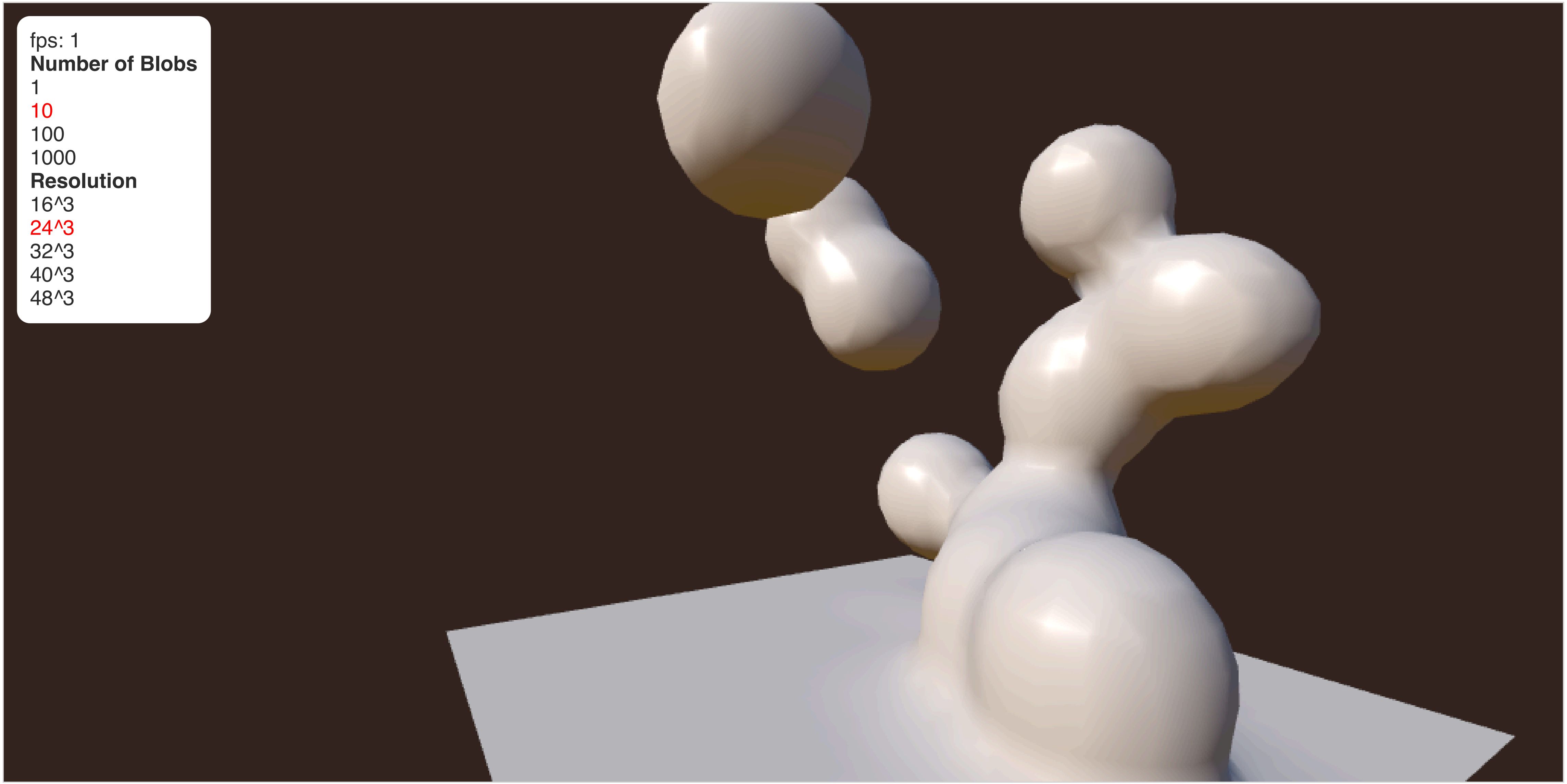
- **Move mouse** around, the particles will be generated following the mouse
- Press **space to pause**, space again to resume
- **Click and drag** to rotate the camera, use scroll to zoom in and out
- Play with the **controls** and let's see what you can find!
    - **Factor:** speed at which the particles move
    - **Evolution:** the variation over time of the particles flow
    - **Rotation:** speed at which the particle field auto-rotates
    - **Radius:** radius of a sphere that repels particles
    - **Pulsate:** if enabled, the sphere pulsates, pushing in and out
    - **ScaleX/Y/Z:** set the scale of each axis of the particles
    - **Scale:** change the overall scale of the particles

Too many? Too litte?
Try almost none, a few, some, regular, quite a few, a lot, INSANE

factor	0.5
evolution	0.5
rotation	0.5
radius	2
pulsate	☐
scaleX	0.1
scaleY	1
scaleZ	5
scale	1

Source

Implicit Surfaces


fps: 1

Number of Blobs

1

10

100

1000

Resolution

16^3

24^3

32^3

40^3

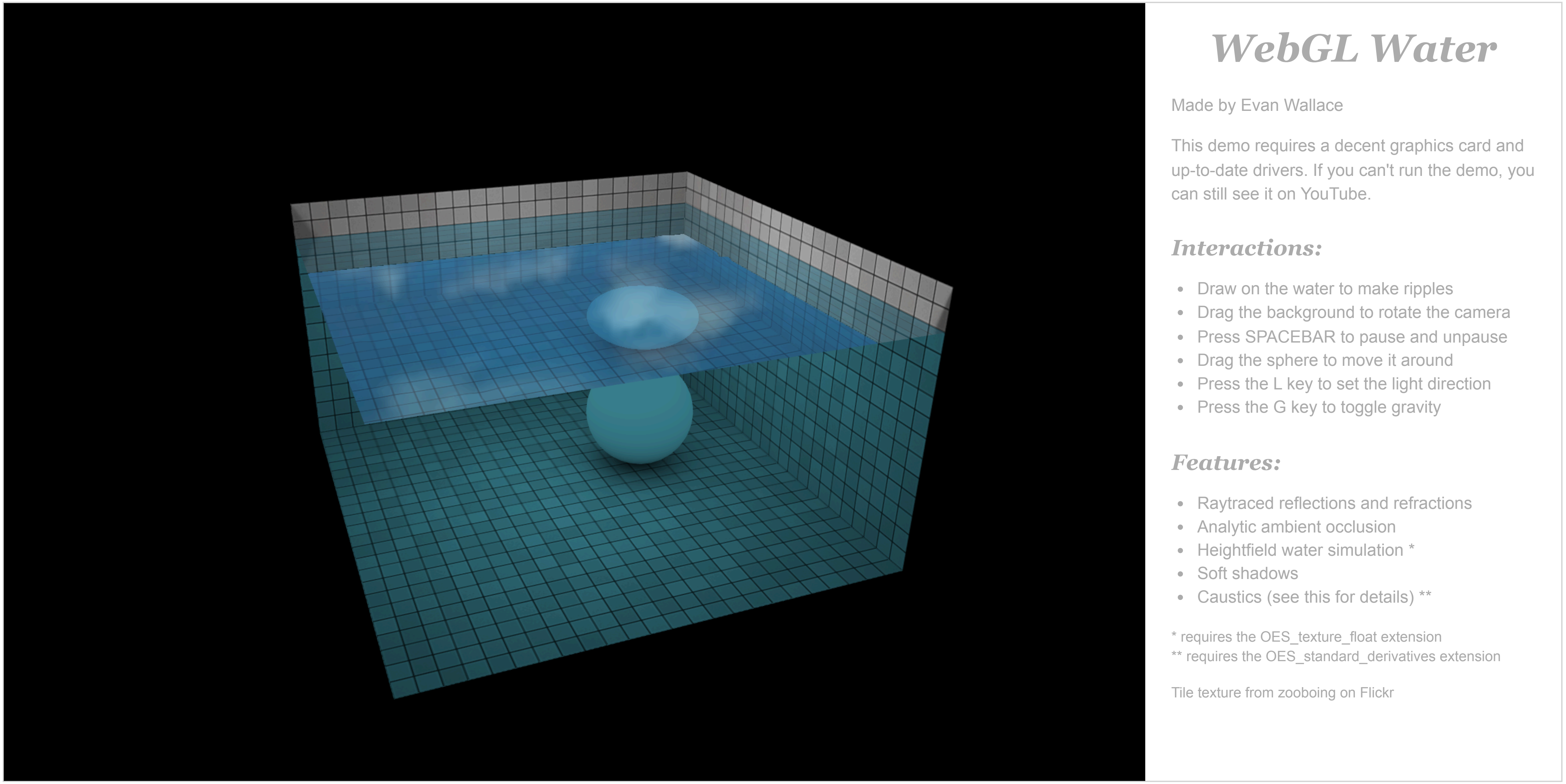
48^3

Source

Games

Source

Raytracing & Fluid Sim



WebGL Water

Made by Evan Wallace

This demo requires a decent graphics card and up-to-date drivers. If you can't run the demo, you can still see it on YouTube.

Interactions:

- Draw on the water to make ripples
- Drag the background to rotate the camera
- Press SPACEBAR to pause and unpause
- Drag the sphere to move it around
- Press the L key to set the light direction
- Press the G key to toggle gravity

Features:

- Raytraced reflections and refractions
- Analytic ambient occlusion
- Heightfield water simulation *
- Soft shadows
- Caustics (see this for details) **

* requires the OES_texture_float extension
** requires the OES_standard_derivatives extension

Tile texture from zooboing on Flickr

Source