

Introduction to Computer Graphics

- Course Presentation -

Joaquim Madeira

February 2023

Overview

- Motivation
- Tentative syllabus
- Evaluation
- Class organization
- Some useful books

MOTIVATION

What is Computer Graphics ?

- Watch the **videos** after the lecture !

What is Computer Graphics ?

- Computer graphics deals with **generating images** with the aid of computers
- **Core technology** in digital photography, **film**, video **games**, cell phone and computer **displays**, **virtual and augmented reality**, and many **specialized applications**

[Wikipedia]

What is Computer Graphics ?

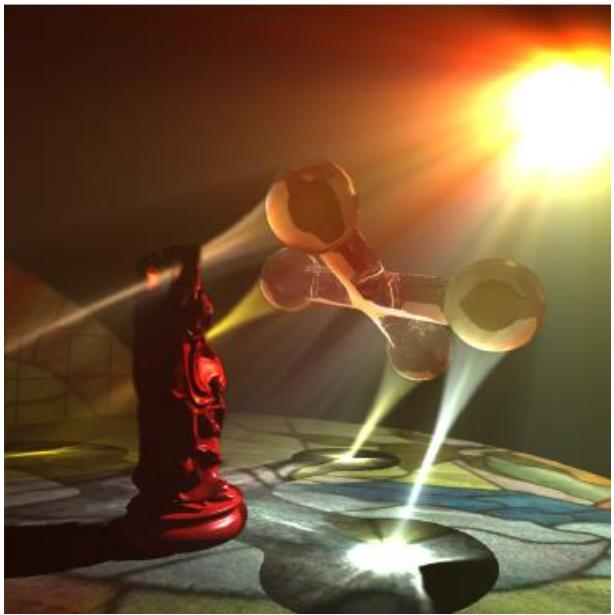
- CG encompasses 3D modeling, rendering, computer animation, data and information visualization, VR/AR, etc.
- CG is responsible for displaying art and image data effectively and meaningfully
- CG development has had a significant impact on many types of media and has revolutionized animation, movies, advertising, video games, etc.

[Wikipedia]

Application Examples



Realistic image synthesis



Global illumination effects

Reflection editing

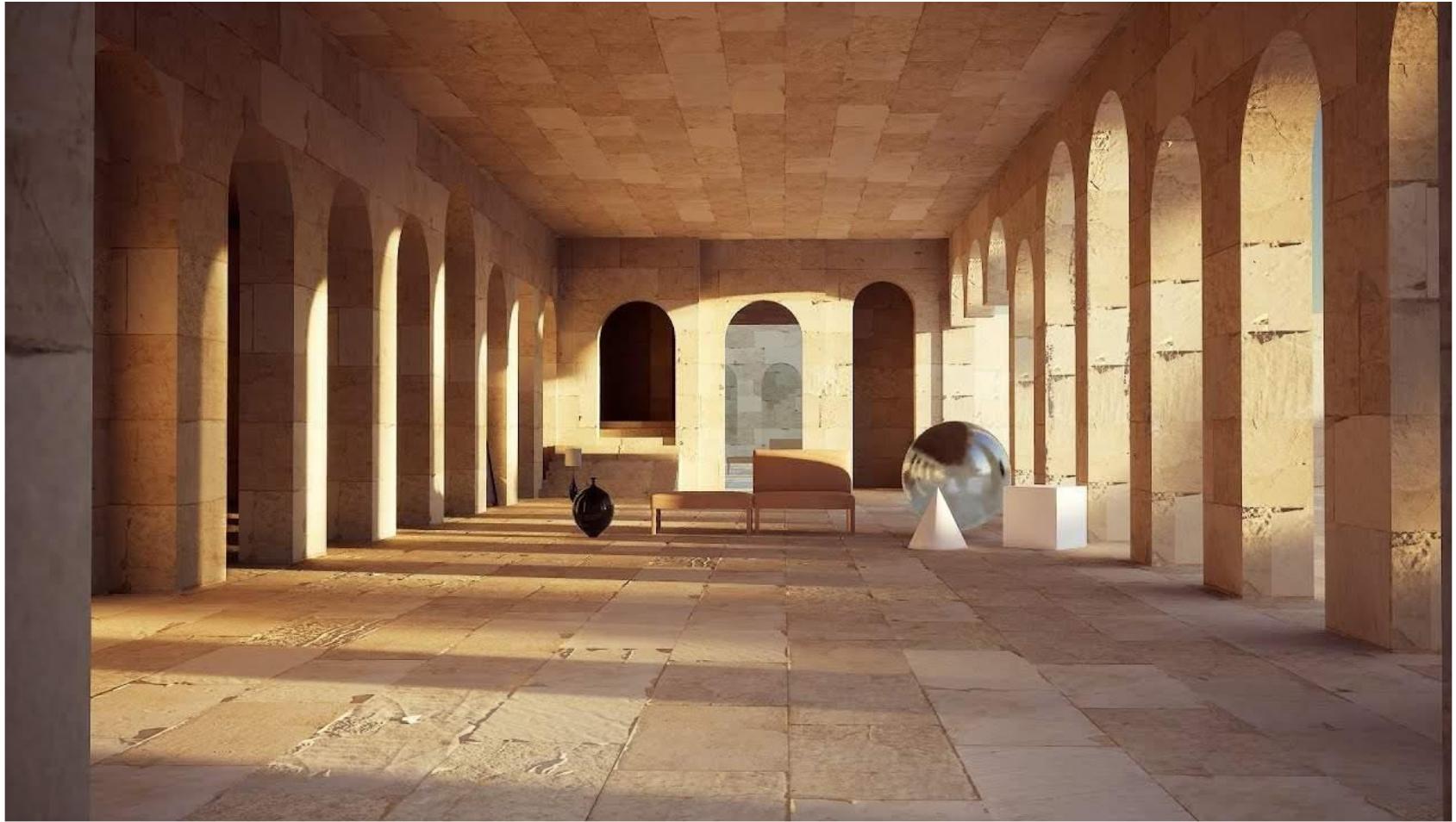


[<http://www.master-visual-computing.de/>]

Realistic Image Synthesis – dezeen



Global Illumination



Application Examples

- From the **real-world** to synthetic **models**



[<http://www.master-visual-computing.de/>]

artec3d.com

■ Professional 3D scanners



artec3d.com – Application areas

- Industrial models
 - Reverse engineering / Quality inspection / ...
- Healthcare
- Science and Education
- Art and Design

Application Examples



Data Visualization

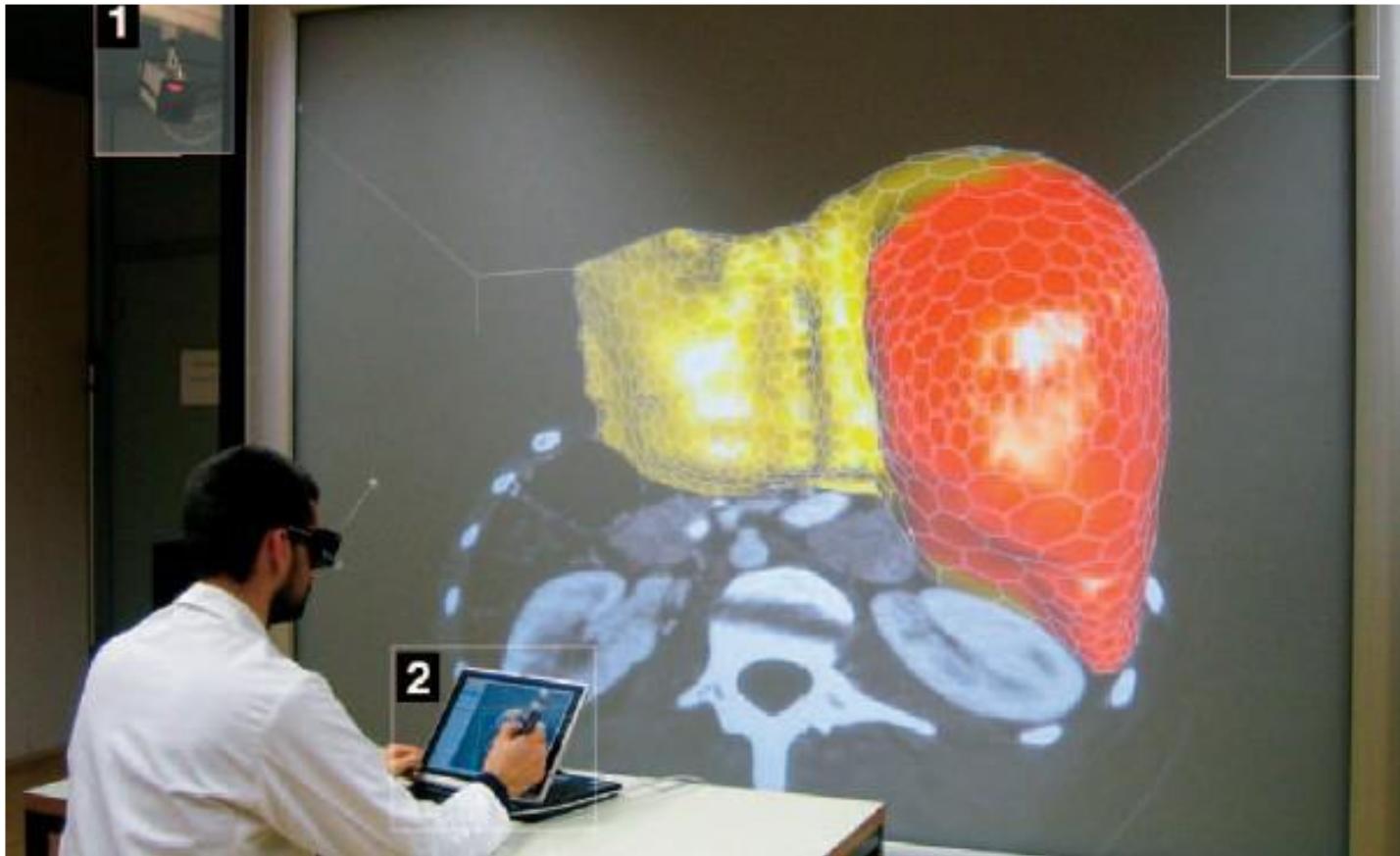


Internet Security

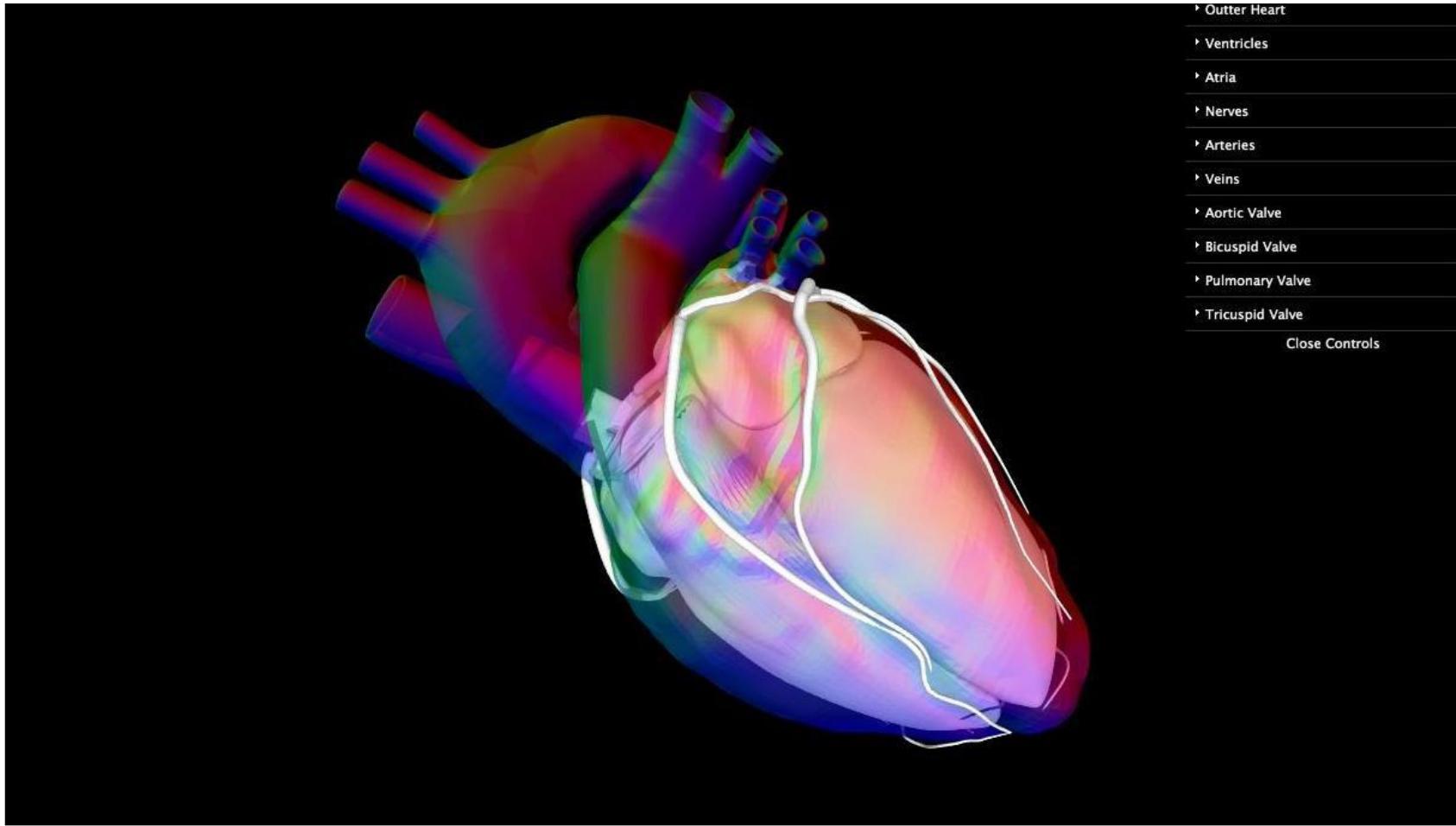


Virtual / Augmented Reality

Medical Visualization



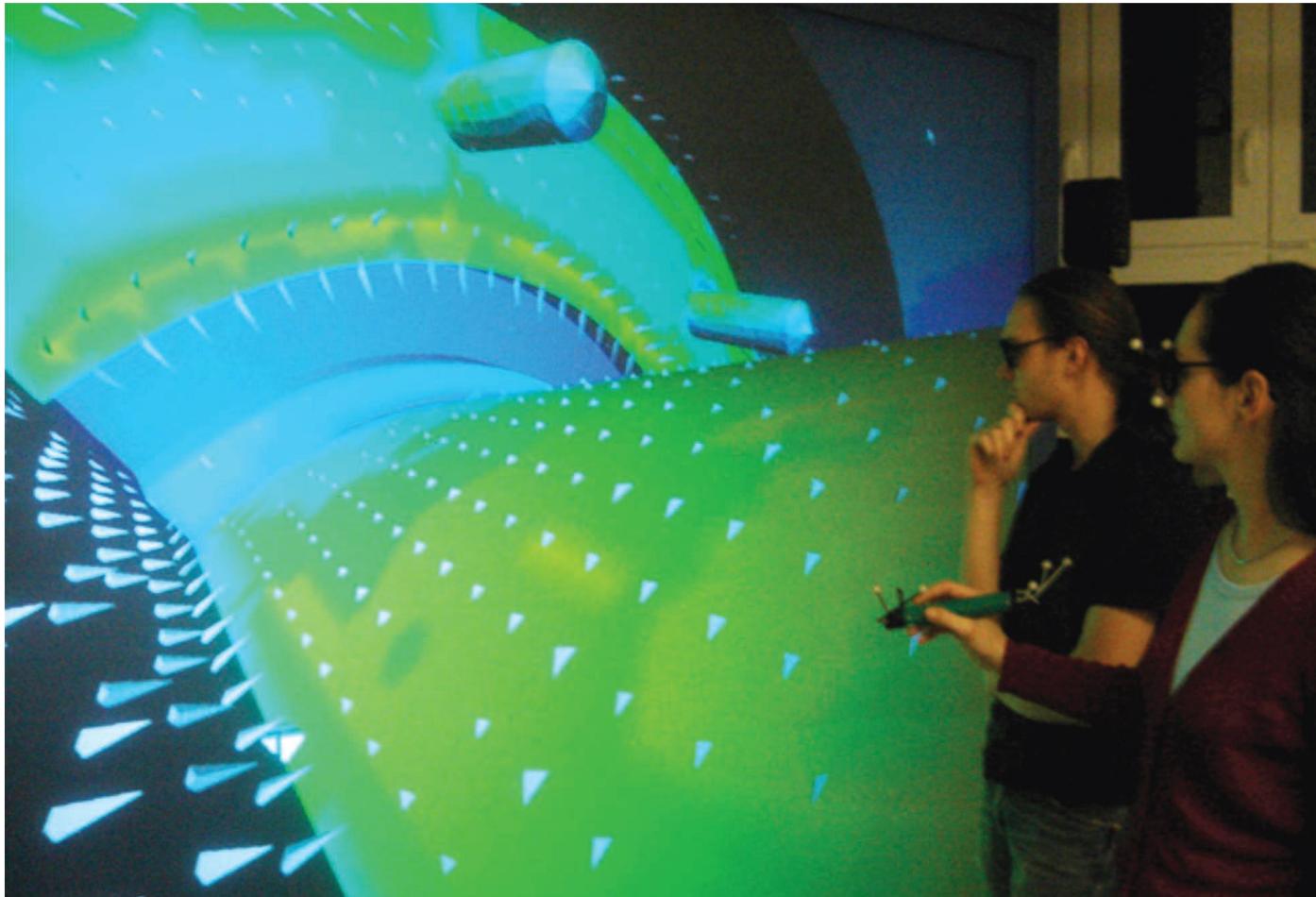
Medical Visualization



Visualizing Population Density



VR / AR Visualization



[Weidlich et al, 2008]

immersivelearning.news

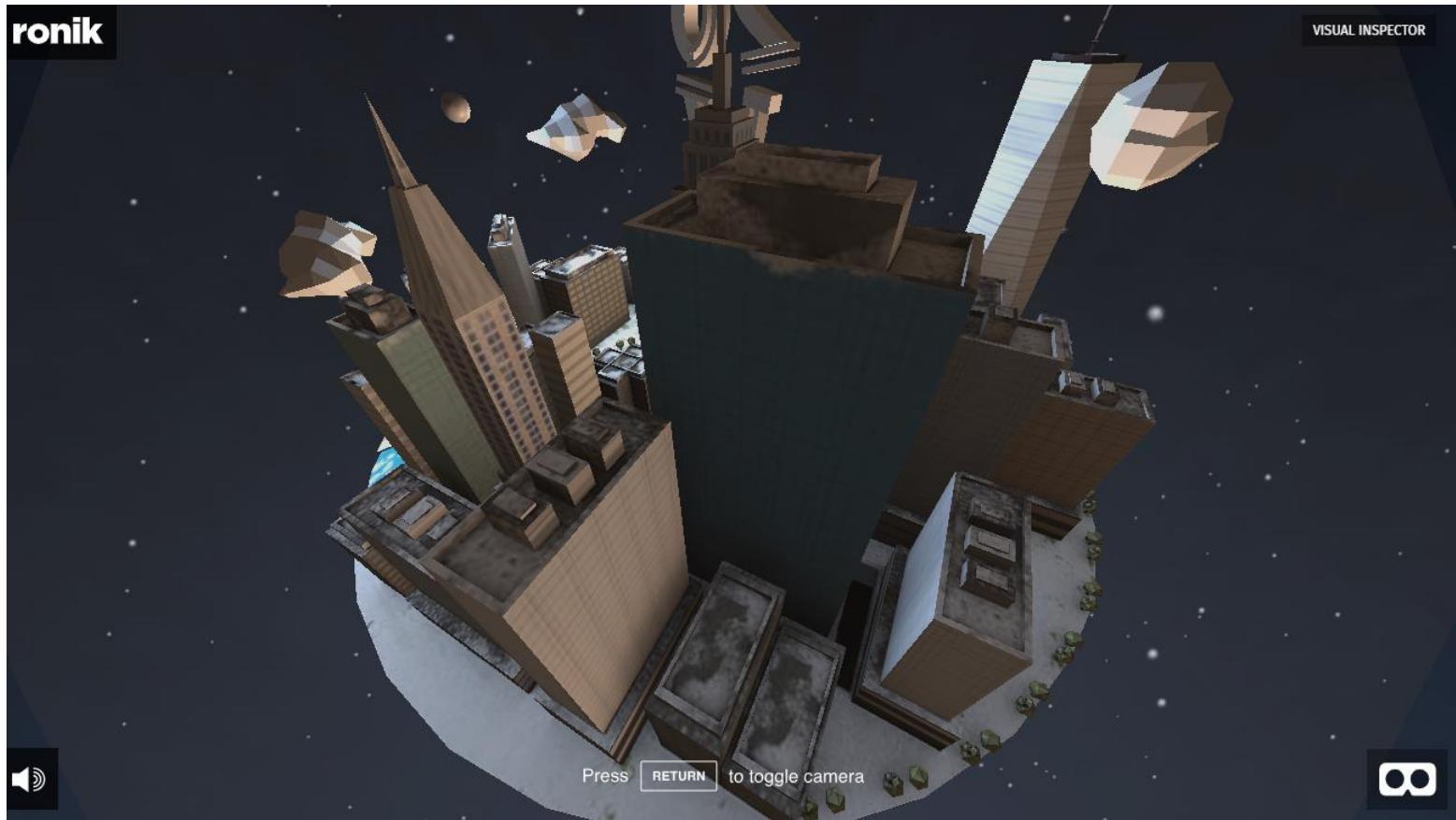


Mobile Graphics



[SIGGRAPH Asia 2017 Course Notes]

Web-based VR



[<https://aframe.io/examples/showcase/snowglobe/>]

Reconstructing Ancient Sculptures



[<https://spectrum.ieee.org/computing/software/ancient-sculptures-return-to-mosul-as-digital-reconstructed-replicas>]

First holographic 5G call in Portugal



Realizada a primeira chamada holográfica 5G em Portugal

14 AGO 2019 · NOTÍCIAS

39 COMENTÁRIOS

A Vodafone e a TVI realizaram a primeira transmissão holográfica 5G em tempo real, um marco inédito no nosso País.

[<https://www.youtube.com/watch?v=9zoC5iT6dXI>]

VR headsets for cows

Russian cows fitted with VR headsets show 'reduced anxiety and improved emotional mood'



A Russian dairy farm has strapped virtual reality headsets to its cows in an experiment to "improve their wellbeing". CREDIT: @OLOLENTACH/TWITTER

NEWS WEBSITE OF THE YEAR

The Telegraph

Coronavirus News Politics

Follow

By **Matthew Bodner**, MOSCOW

26 NOVEMBER 2019 • 7:37PM

AR goggles for dogs

The US military is trialing augmented reality goggles for dogs

By Sara Spary, CNN Business

Updated 1623 GMT (0023 HKT) October 9, 2020



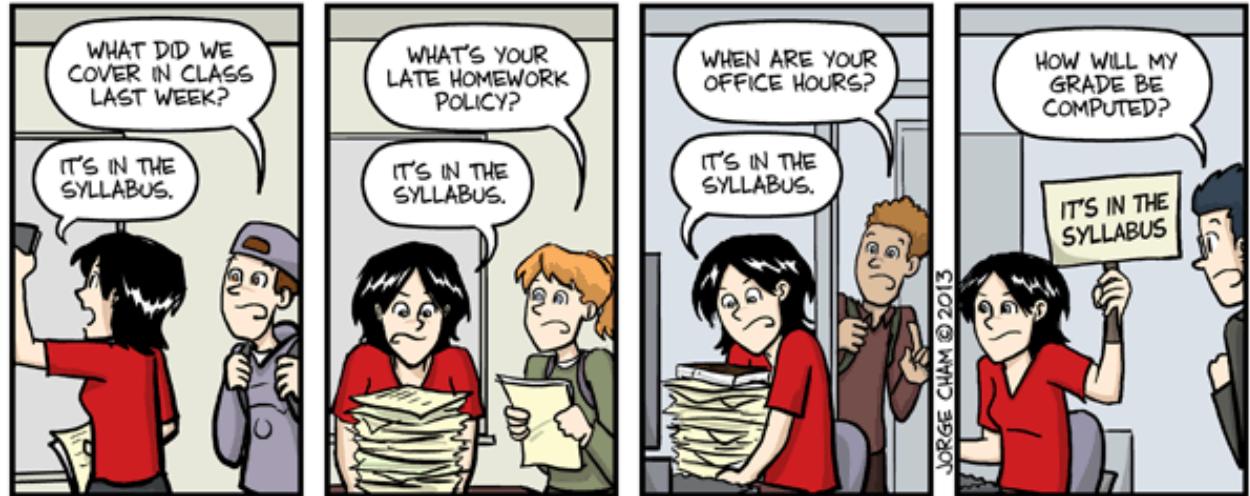
FROM COMMAND SIGHT/US ARMY

The technology, which the US Army says is the first of its kind, works by letting a handler see everything the dog can see and then provide specific commands using visual cues that show up in the dog's line of vision.

[<https://edition.cnn.com/2020/10/09/tech/army-dogs-goggles-scli-intl/index.html>]

Computer Graphics vs ...

		Output	
		Model	Image
Input	Model	Geometric Modeling	Computer Graphics
	Image	Computer Vision	Image Processing



[phdcomics.com]

SYLLABUS

Tentative planning

- Introduction to CG and Three.js
- The Visualization Pipeline
- 2D and 3D Transformations. Projections
- Illumination and Shading
- 3D Modeling using Triangle Meshes
- 1st Project Presentations
- Textures
- Animation
- Advanced Techniques
- 2nd Project Presentations



[irinstitutes.org]

EVALUATION

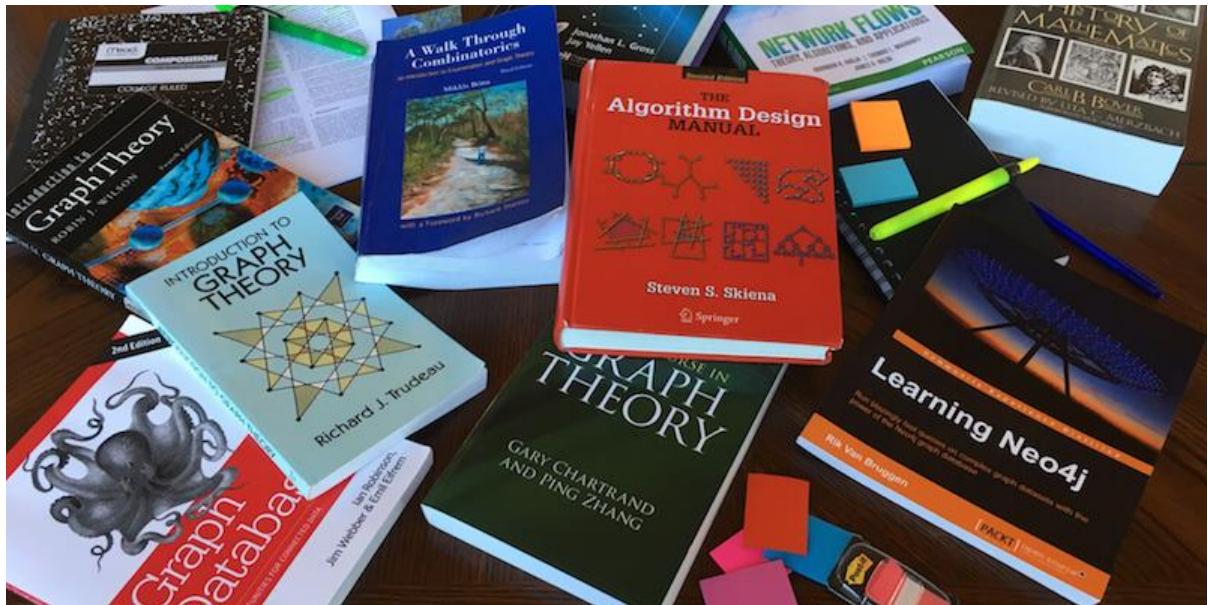
Grading

- Mixed grading / Avaliação discreta
 - Exam (“normal” or “recurso”) : 35 %
 - 1 project : 60%
 - Class participation : 5%
- 1 project using Three.js
 - Individual – not group work !
 - Intermediate + Final delivery
 - Code + video + slides + presentation + live demo

ORGANIZATION

Lecture Organization

- Additional materials: videos, etc.
 - View them before or after the lecture !
- 1st part : Lecture / presentation
- 2nd part : Hands-on : Design / programming /...
- Weekly OT on Zoom
 - Mondays : 18:00 to 19:00



[hackernoon.com]

USEFUL BOOKS

Reference books

Edward Angel, Dave Shreiner

Interactive Computer Graphics, 7th Ed.

Addison-Wesley, 2015

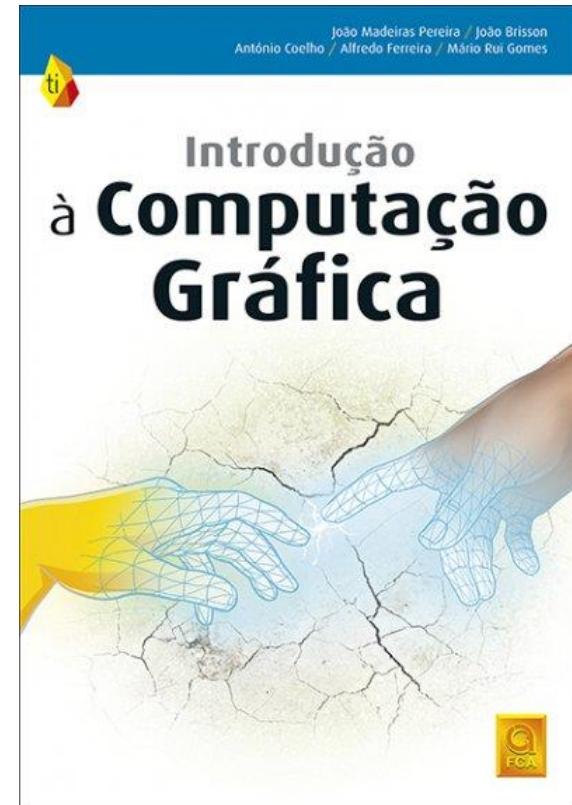
Jos Dirksen

Learning Three.js , 2nd Ed.

Packt Publishing, 2015

Um livro recente

J M Pereira, J Brisson, A Coelho, A Ferreira e M R Gomes
Introdução à Computação Gráfica.
FCA, 2018



Additional books

- Hughes, J. F., A. van Dam, et al., *Computer Graphics – Principles and Practice*, 3rd ed., Addison-Wesley, 2014
- Foley, J., A. van Dam, et al., *Introduction to Computer Graphics*, Addison Wesley, 1993
- Rogers, D., J. Adams, *Mathematical Elements for Computer Graphics*, 2nd ed., McGraw-Hill, 1989
- ...

Course materials on-line

- CS 123 – Introduction to Computer Graphics
 - Brown University, USA – Andy van Dam
- CS 581 – Graphics Programming
 - Philipps-Universität Marburg, Germany
- ...

Udacity free course

■ Interactive 3D Graphics

- Eric Haines and Gundega Dekena
- Autodesk

SIGGRAPH University on YouTube

- Fundamentals Seminar
- Introduction to 3D Computer Graphics
- Introduction to "Physically Based Shading in Theory and Practice"

Warnings

- Classes are not mandatory !
 - But you should always be present...
- PACO : choose how you want to be graded...

An Introduction to Computer Graphics

Joaquim Madeira

March 2022

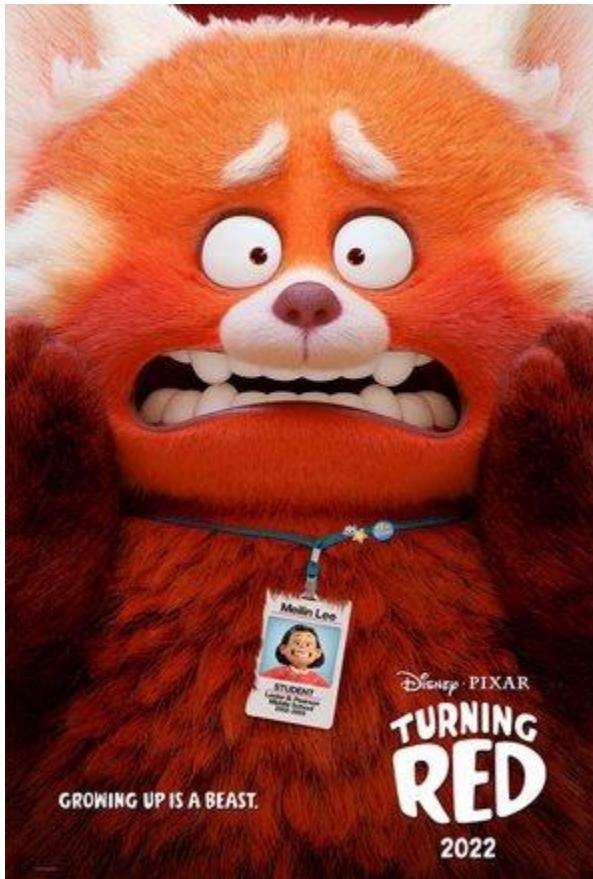
Overview

- What is CG ?
- Application areas
- Evolution and trends
- Main tasks
- CG APIs



PIXAR'S COMPUTER-ANIMATED FILMS

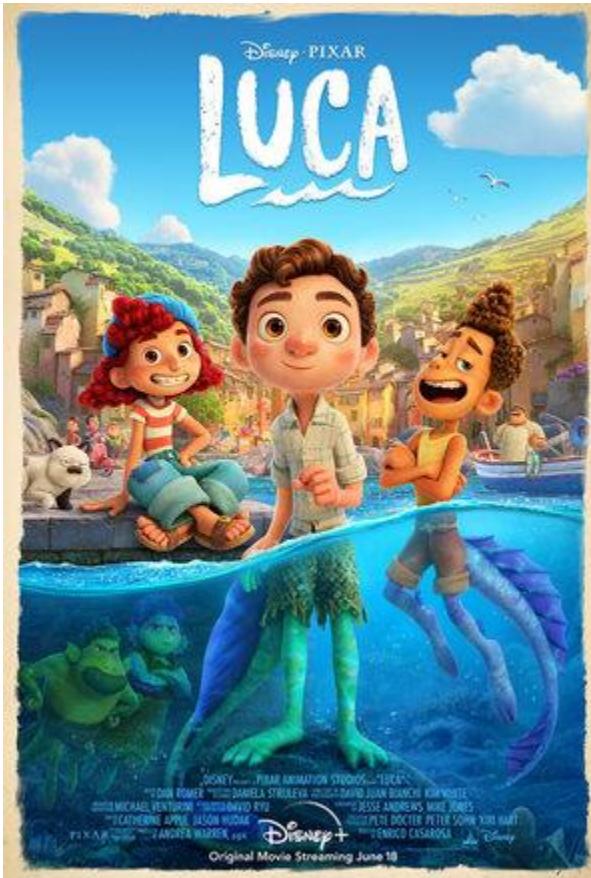
Turning Red (2022)



Trailer at
YouTube

- Pixar's latest computer-animated film

Luca (2021)



Trailer at
YouTube

- Pixar's 2021 computer-animated film

Soul (2020)



[Trailer at
YouTube](#)

- Pixar's 2020 computer-animated film

Onward (2020)



[Trailer at
YouTube](#)

- Pixar's 2020 computer-animated film

Toy Story 4 (2019)



Trailer at
YouTube

- Pixar's 2019 computer-animated film

Incredibles 2 (2018)



Trailer at
YouTube

- Pixar's 2018 computer-animated film

Toy Story (1995)



Trailer at
YouTube

- Pixar's first **entirely** computer-animated feature film

Pixar's Luxo Jr. (1986)



[Trailer at YouTube](#)

- Pixar's second computer-animated short-film

WHAT IS COMPUTER GRAPHICS?

What is Computer Graphics?

- The technology with which **pictures**, in the broadest sense of the word, are
 - Captured or generated, and presented
 - Manipulated and / or processed
 - Merged with other, non-graphical application data
- It includes:
 - Integration with other kinds of data – **Multimedia**
 - Advanced dialogue and **interactive technologies**

[CG Topics – Darmstadt]

What is Computer Graphics?

- Computer graphics generally means **creation**, **storage** and **manipulation** of **models** and **images**
- Such models come from a diverse and expanding set of fields
 - Physical, biological, mathematical, artistic, and conceptual / abstract structures

[Andy van Dam]

What is Computer Graphics?

- Computer Graphics deals with all aspects of **creating images with a computer**
 - Hardware
 - Software
 - Applications
- How was this image produced?



[Angel]

APPLICATION AREAS

CG – Application areas

- Entertainment
 - Computer games
 - Animation films
 - Special effects
- Engineering / Architecture
 - Computer-Aided Design (CAD)
 - Data Visualization
- Medicine
 - Visualization
- ...

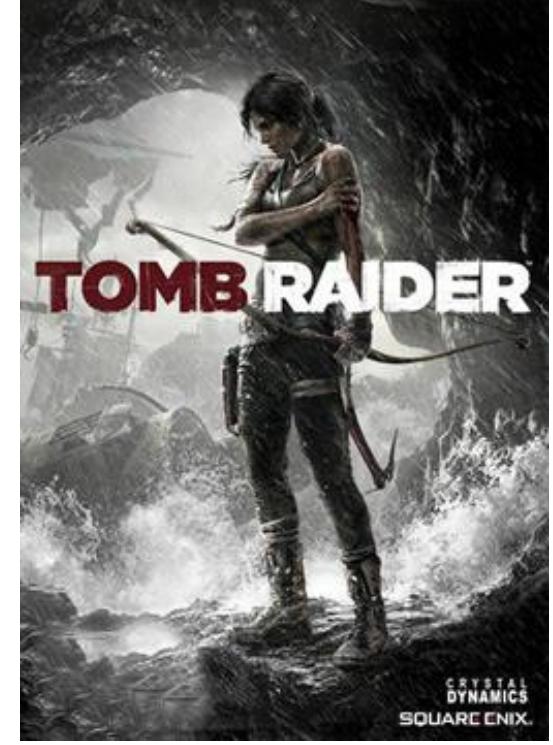
Games – *Lara Croft*



1996



2007



2013

[Wikipedia]

The Games With The Most Impressive Graphics, Ranked

Graphics can make a good game look great. Regardless of the actual gameplay, here are the games with the most undeniably impressive graphics

BY JAMES CARR

UPDATED FEB 05, 2022



<https://gamerant.com/video-games-best-graphics/>

10 Older Video Games That Still Boast Amazing Graphics

Despite how old some of these classic games are, their graphics still stand the test of time and were amazing in their day. These are the standouts.

BY DEREK DRAVEN

PUBLISHED APR 22, 2021



<https://screenrant.com/classic-older-video-games-that-best-graphics/>

10 Games With The Best Graphics Ever (At Their Time Of Release)

Gorgeous graphics tend to make an average game ten times better, and these games in particular wowed players with incredible visuals.

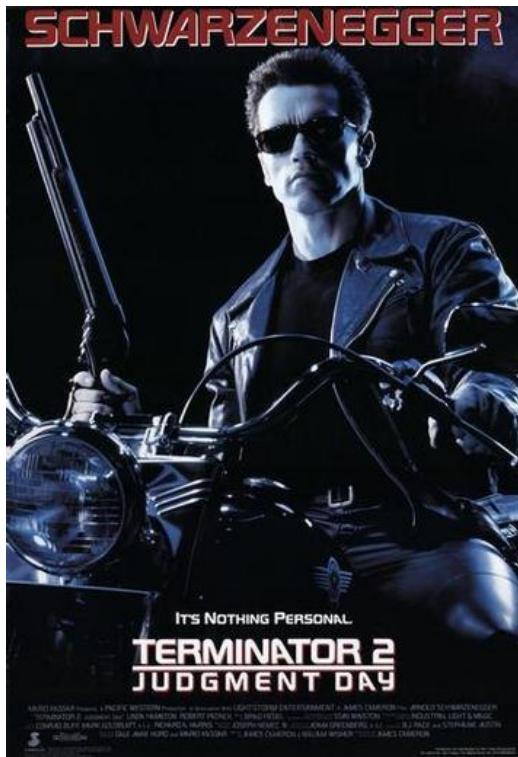
BY GAVIN FRASER MACKENZIE

PUBLISHED SEP 07, 2021



<https://gamerant.com/best-game-graphics-at-release/>

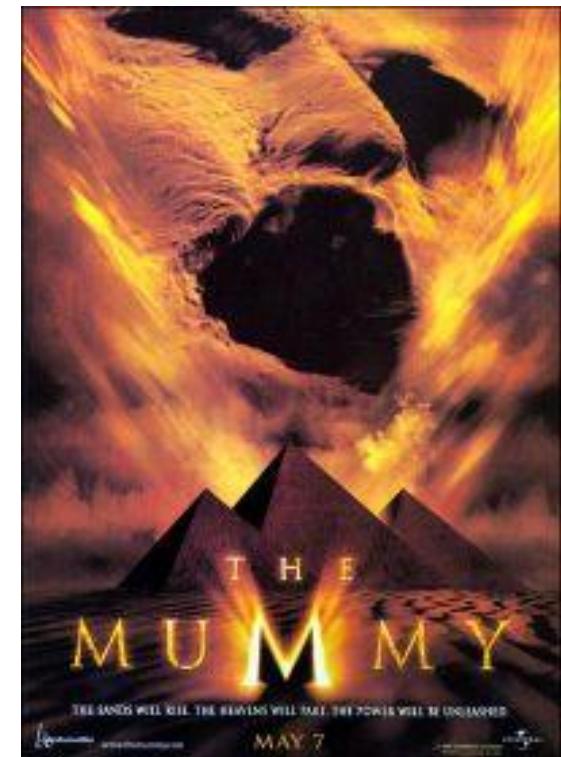
Special effects – ILM



1991



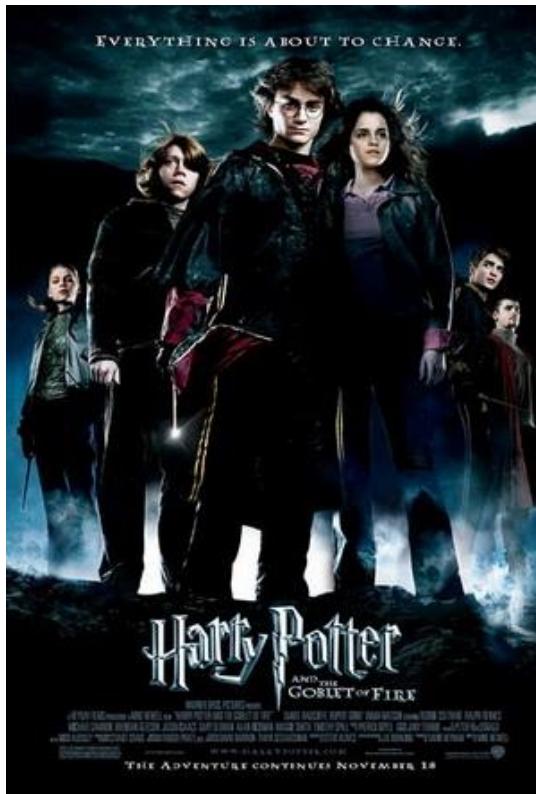
1994



1999

[Wikipedia]

Special effects – ILM



2005



2009

[Wikipedia]



2013

Special effects – ILM



2015



2016

[Wikipedia]



2017

Special effects – ILM



2018



2019



2019

[Wikipedia]

14 groundbreaking movies that took special effects to new levels

Lucien Formichella Jan 11, 2020, 6:30 PM



<https://www.insider.com/most-groundbreaking-cgi-movies-ever-created-2020-1>

35 greatest CGI movie moments of all time

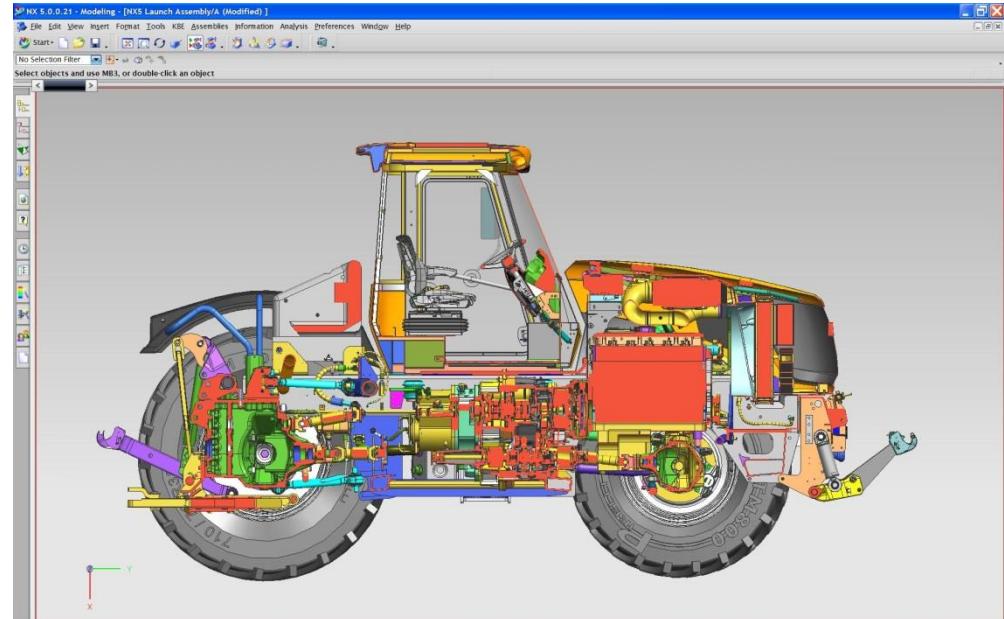
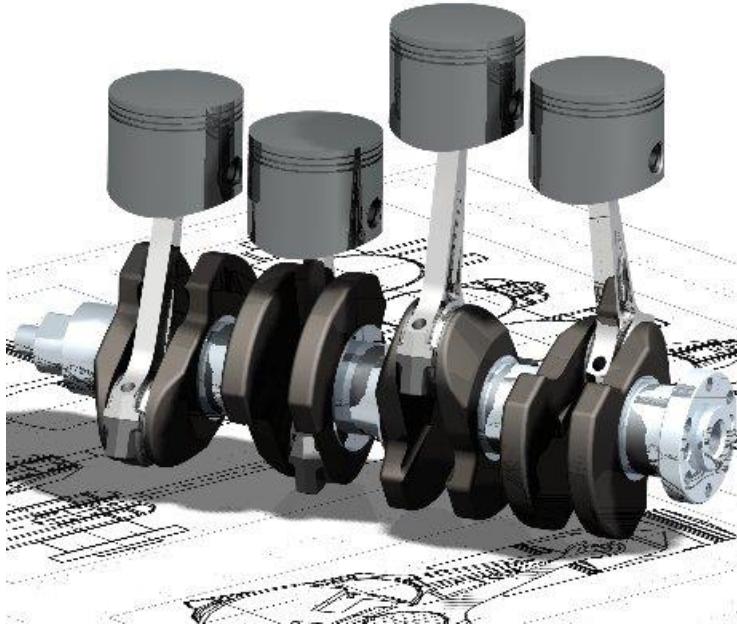
By Creative Bloq Staff published August 01, 2019

Our pick of the best CGI movie moments in live action films.



<https://www.creativebloq.com/3d-tips/cgi-movie-moments-1234014>

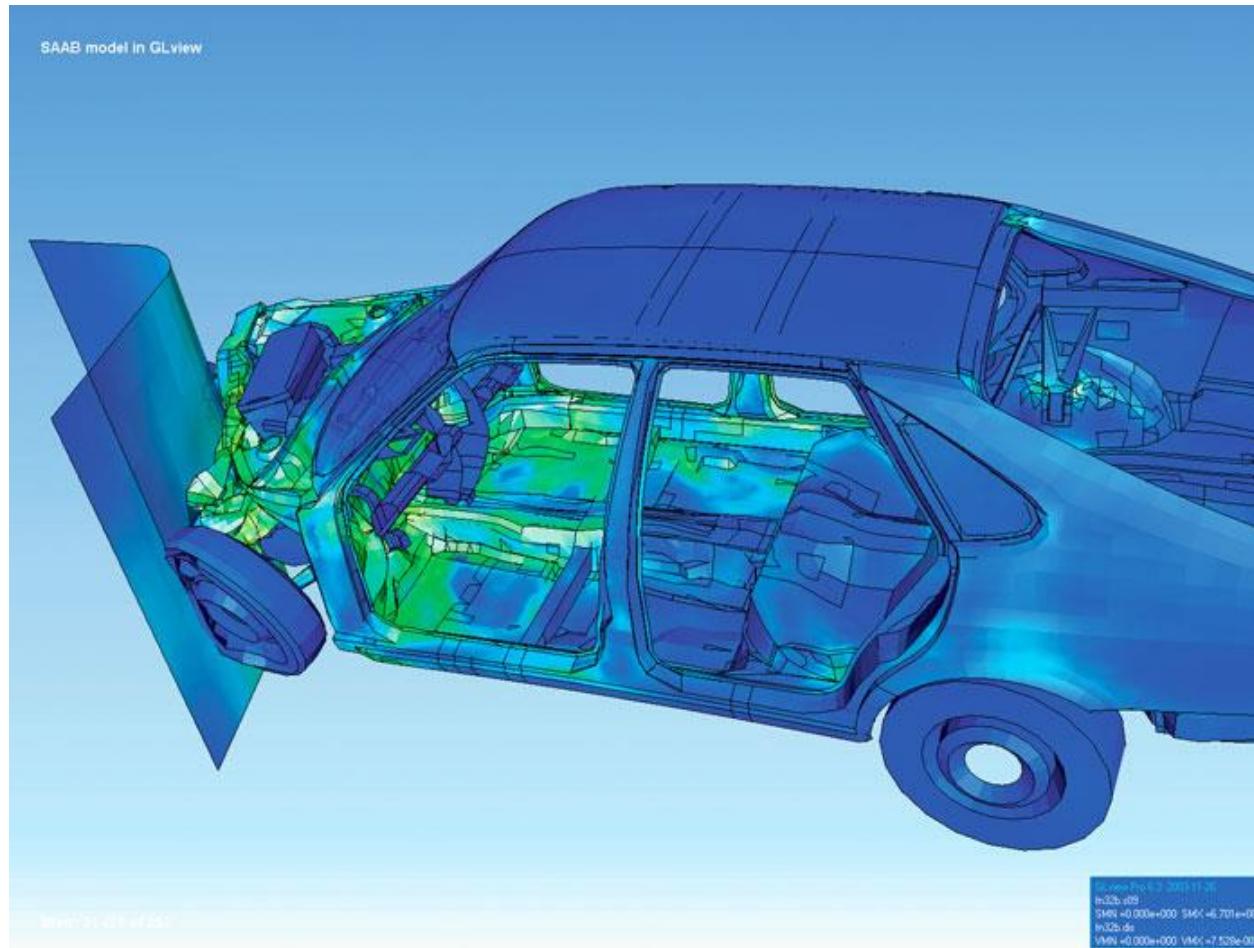
Computer-Aided Design



CAD mockup

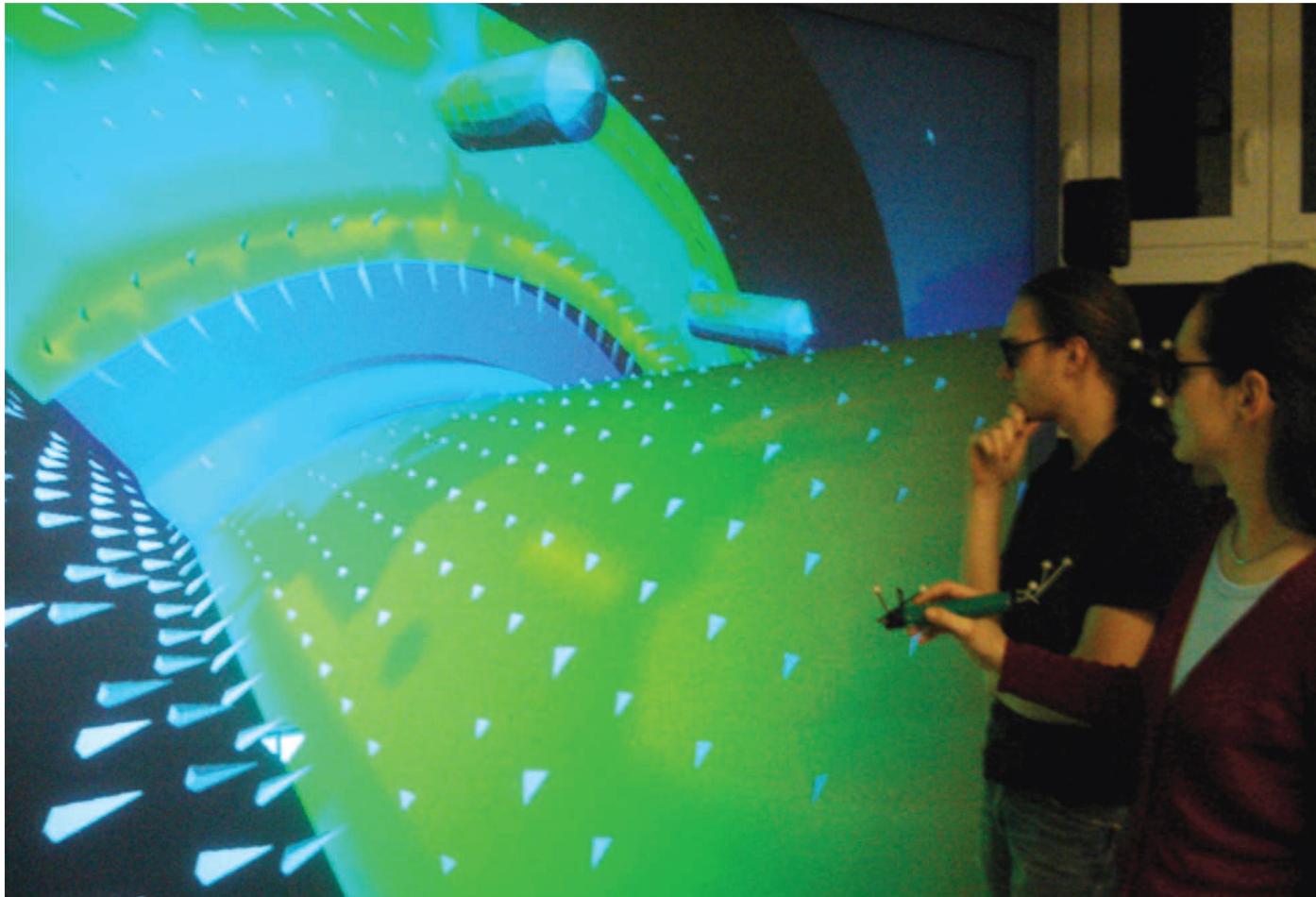
[Wikipedia]

CAD – Simulation



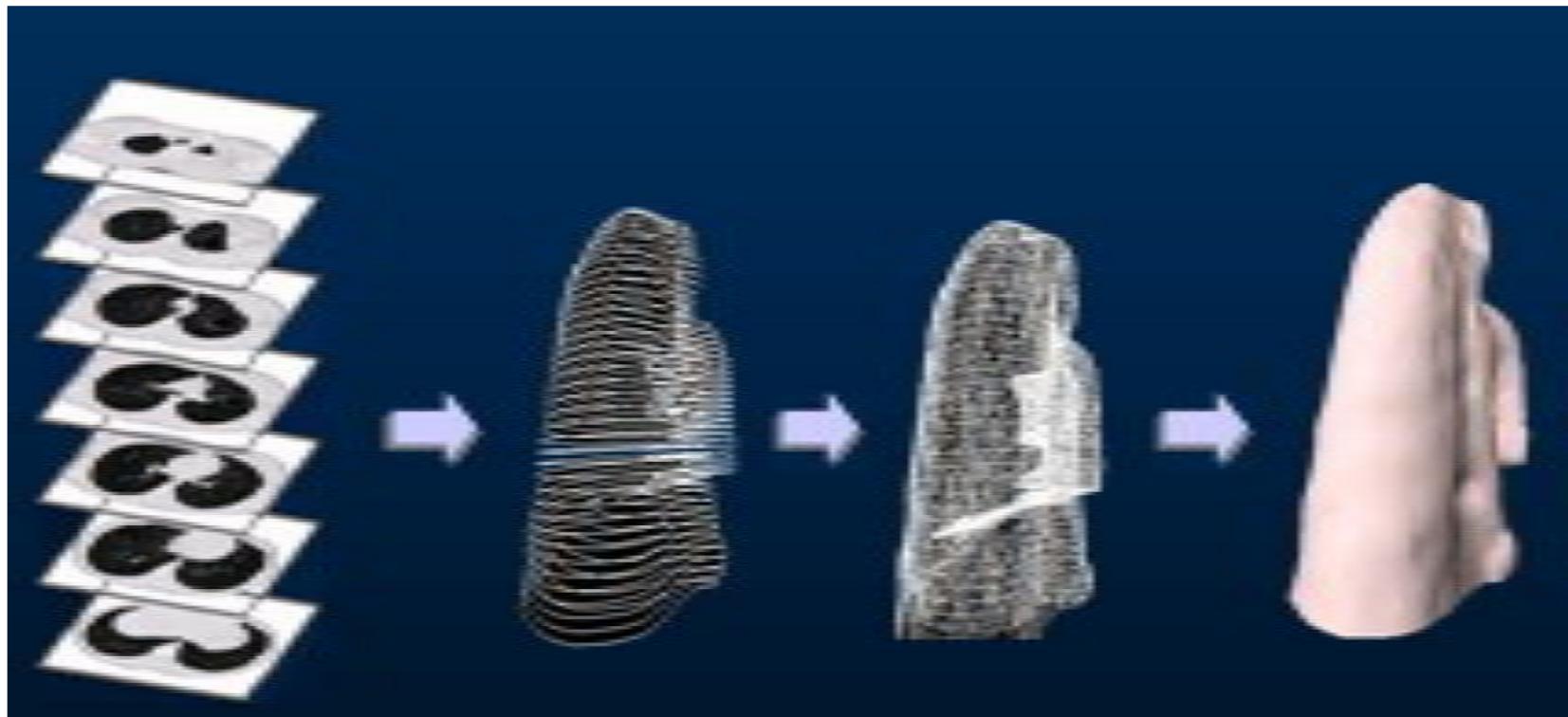
[Wikipedia]

VR / AR Visualization



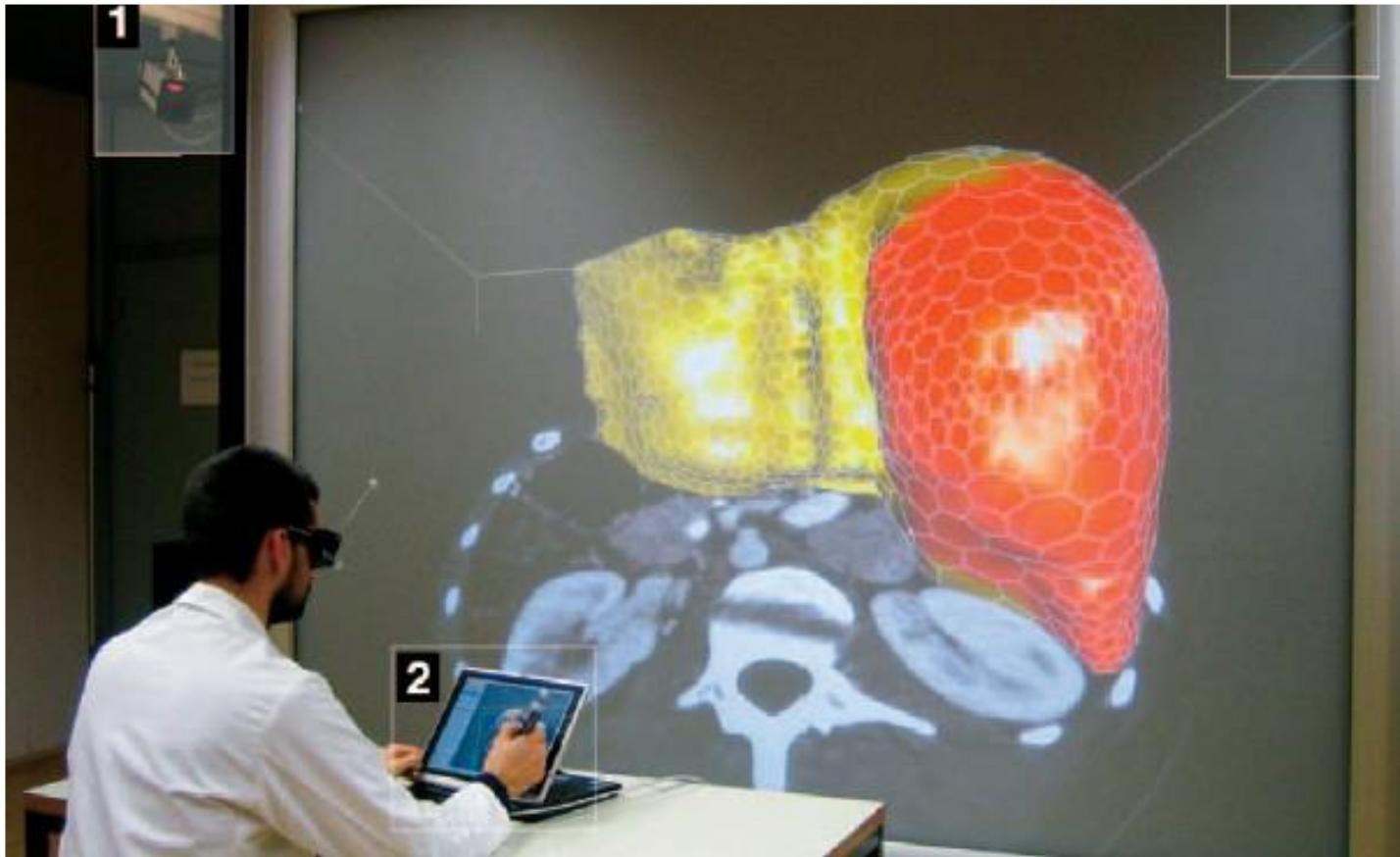
[Weidlich et al, 2008]

Medical Data Visualization

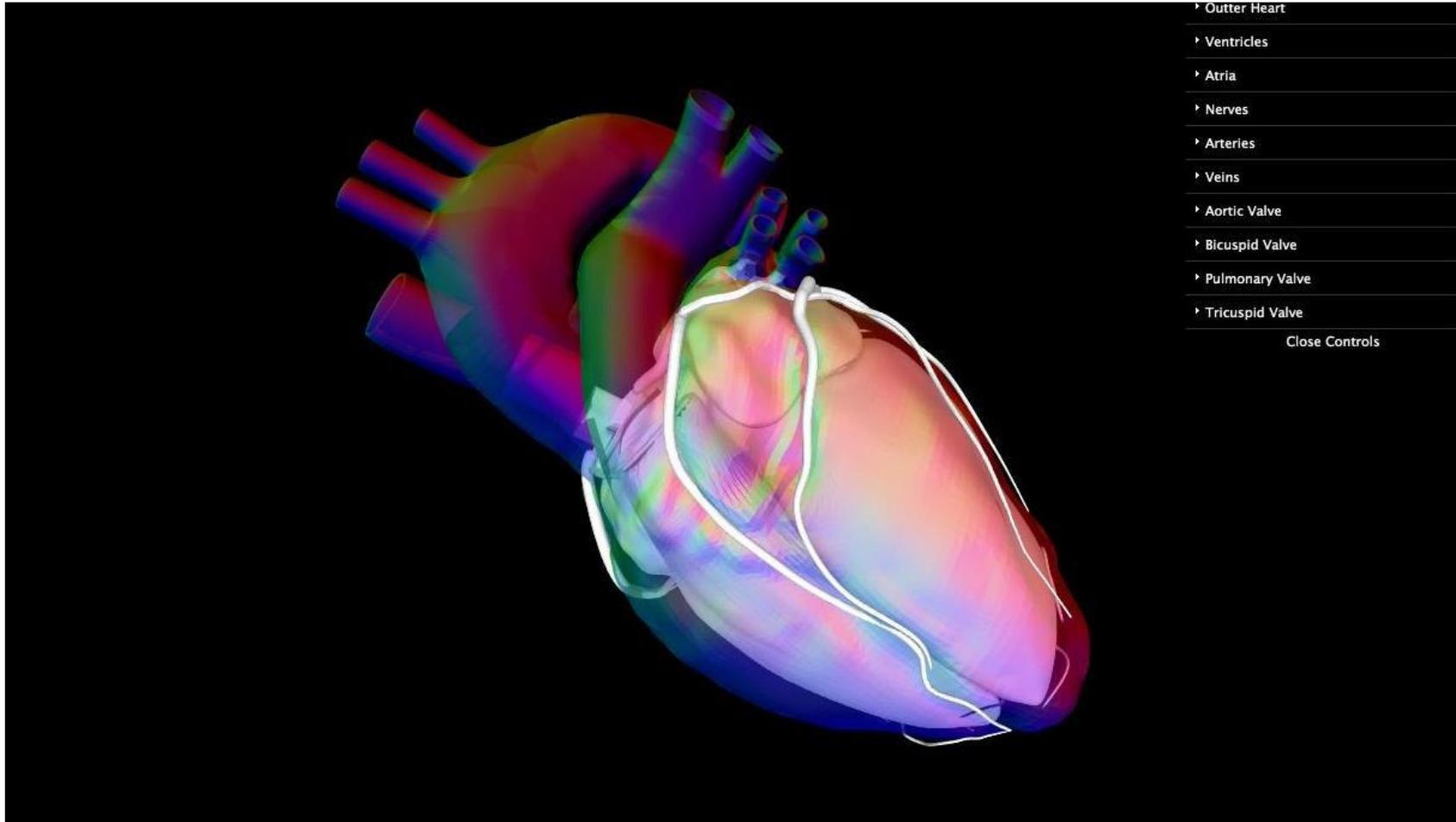


From CT data to lung models
U.Aveiro, 2004

Medical Data Visualization



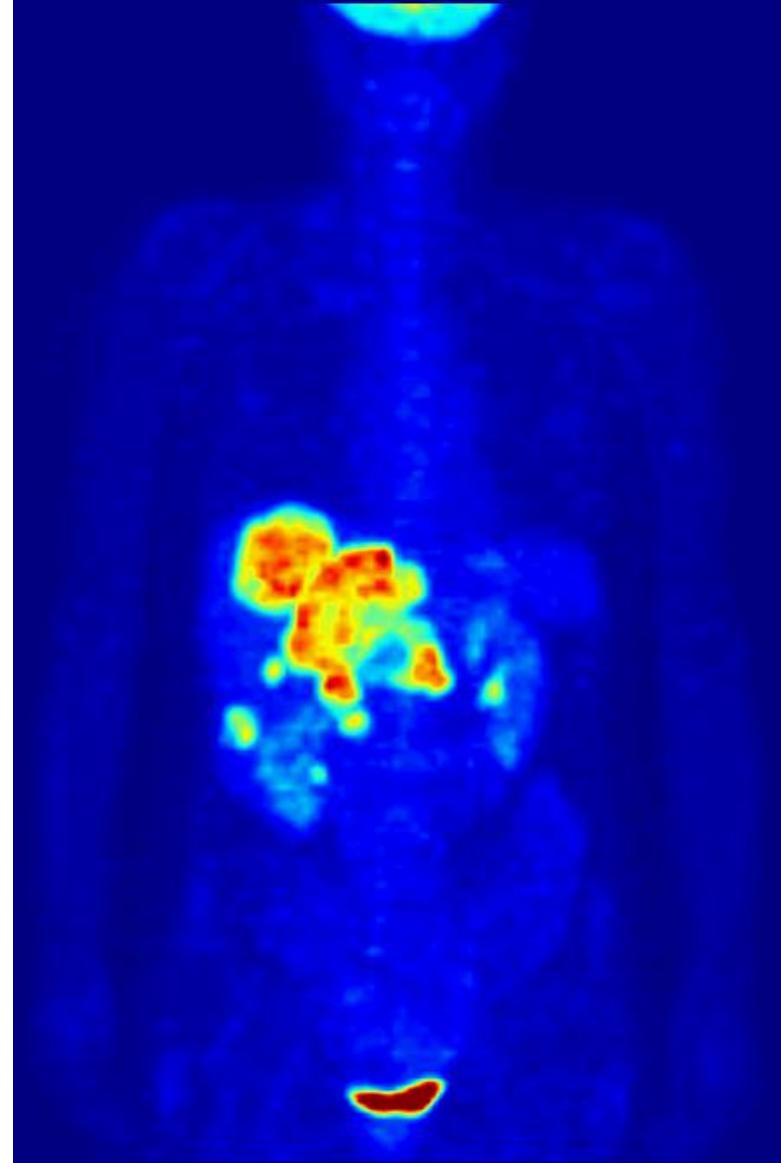
Medical Data Visualization



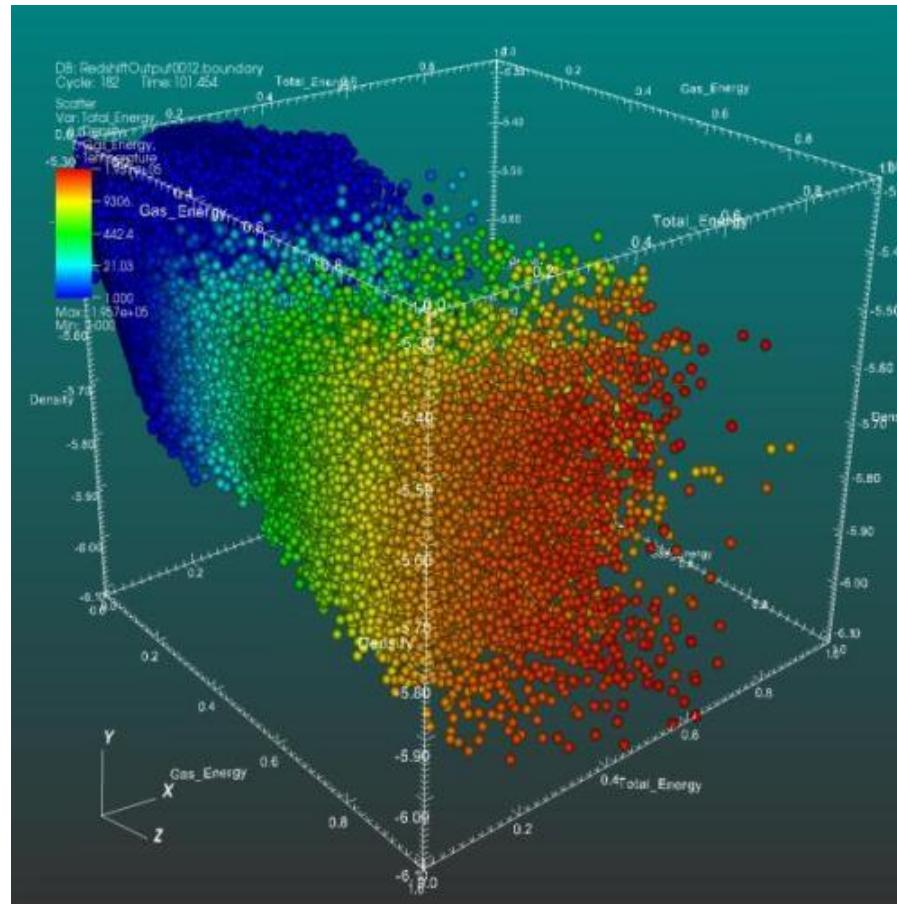
Medical Data Visualization

- PET scan for tumor diagnosis

[Wikipedia]



Data Visualization



[Wikipedia]

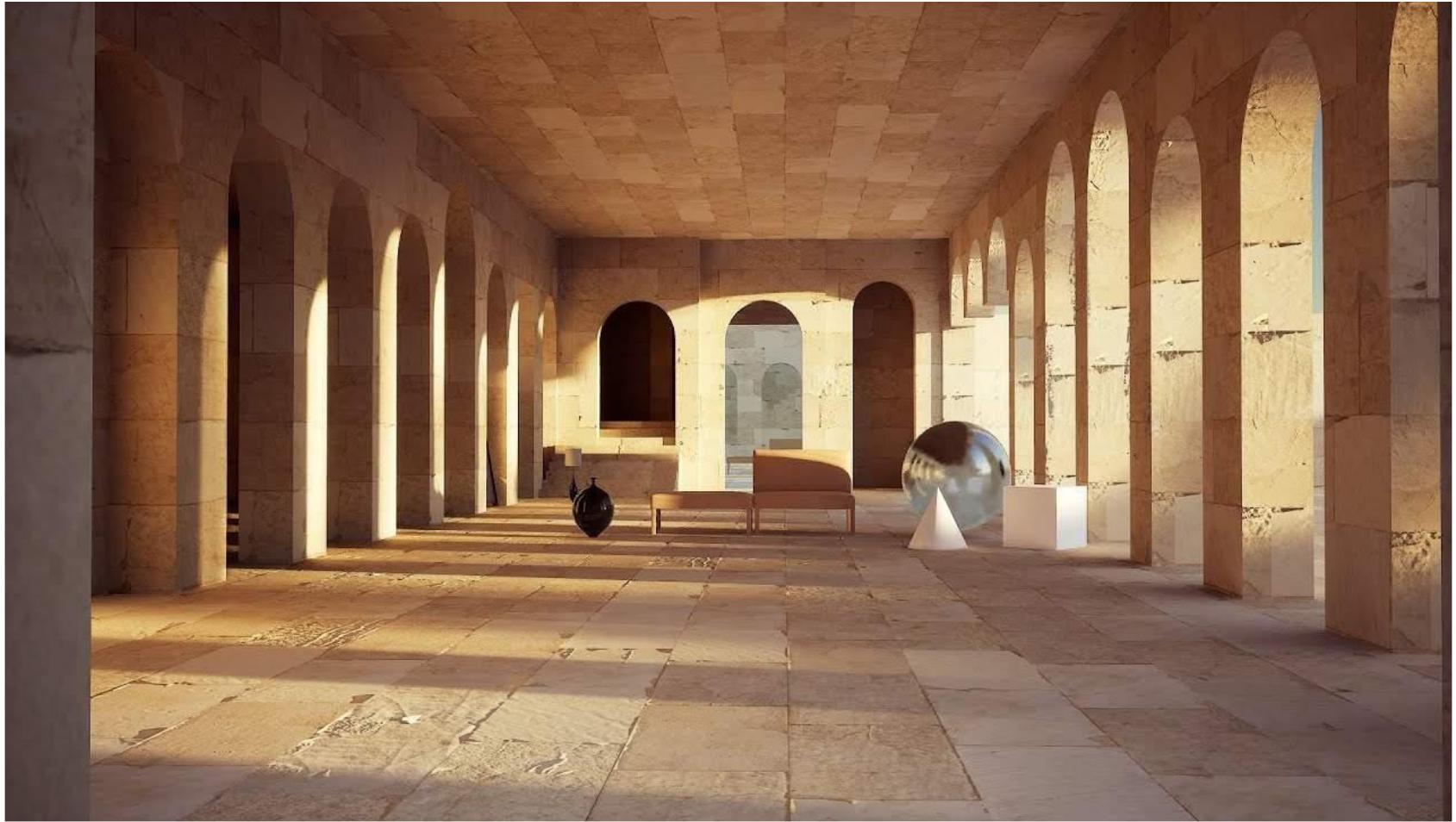
Visualizing Population Density



Realistic Image Synthesis – dezeen



Global Illumination



immersivelearning.news



CG – Some YouTube videos

- SIGGRAPH 2021
 - Technical Papers Trailer
 - VR Theater
 - Computer Animation Festival Electronic Theater



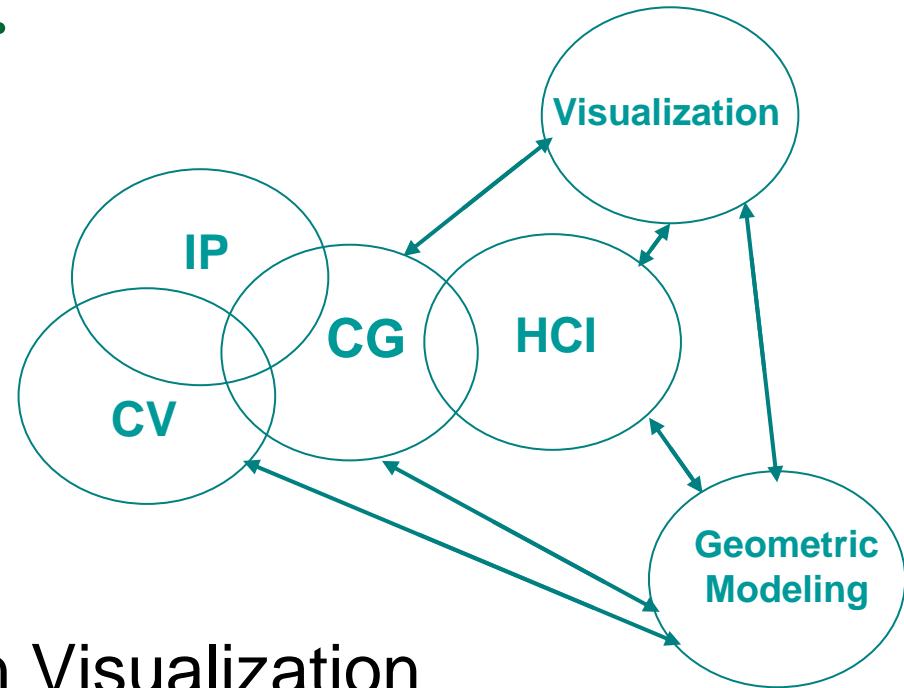
CG IS NOT ALONE

Computer Graphics vs ...

		Output	
		Model	Image
Input	Model	Geometric Modeling	Computer Graphics
	Image	Computer Vision	Image Processing

CG is not alone...

- Core areas:
 - CG, IP, CV and HCI
- Satellite areas:
 - Geometric Modeling
 - Data and Information Visualization
- What is common?
 - CG, IP : image file formats, color models, ...
 - CG, CV : 3D model representations, ...
 - IP, CV : noise removal, filters, ...



Example – Medical Imaging

- Processing pipeline
 - Noise removal
 - Segmentation
 - Generating 2D / 3D models
 - Data visualization
 - User interaction
 - ...



[www.mevislab.de]

EVOLUTION

Lara Croft



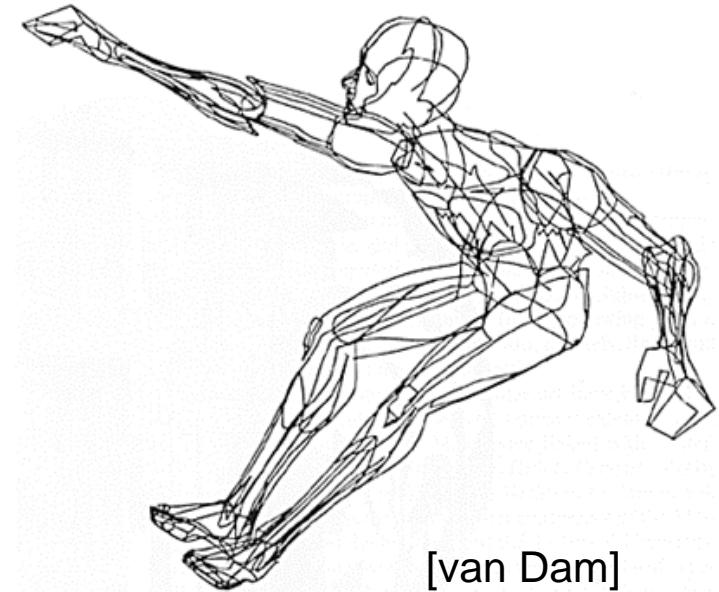
Computer Graphics: 1950 – 1960

■ Earliest days of computing

- Pen **plotters**
- Simple **calligraphic displays**

■ Issues

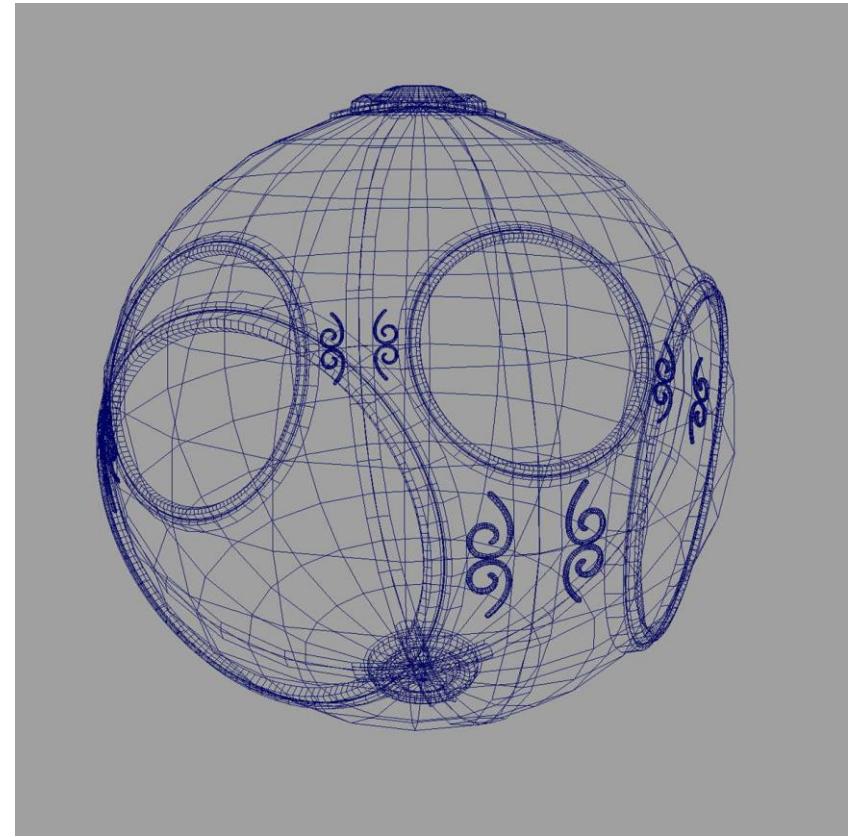
- Cost of display refresh
- Slow, unreliable, expensive computers



[van Dam]

Computer Graphics: 1960 – 1970

- **Wireframe graphics**
 - Draw only lines !



[Angel]

Computer Graphics: 1960 – 1970

■ Ivan Sutherland's Sketchpad

- PhD thesis at MIT (1963)
- Man-machine interaction
- Processing loop
 - Display something
 - Wait for user input
 - Generate new display

Demo at YouTube

[<http://history-computer.com>]

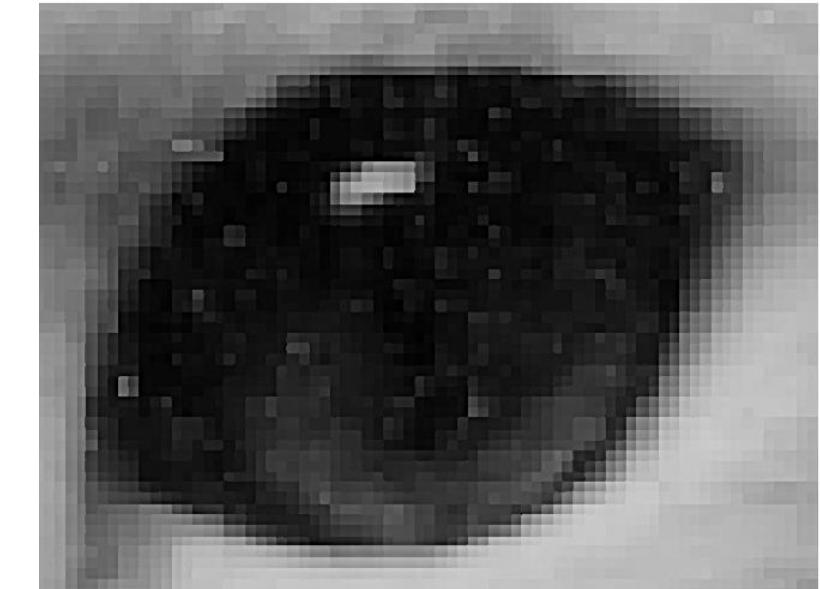
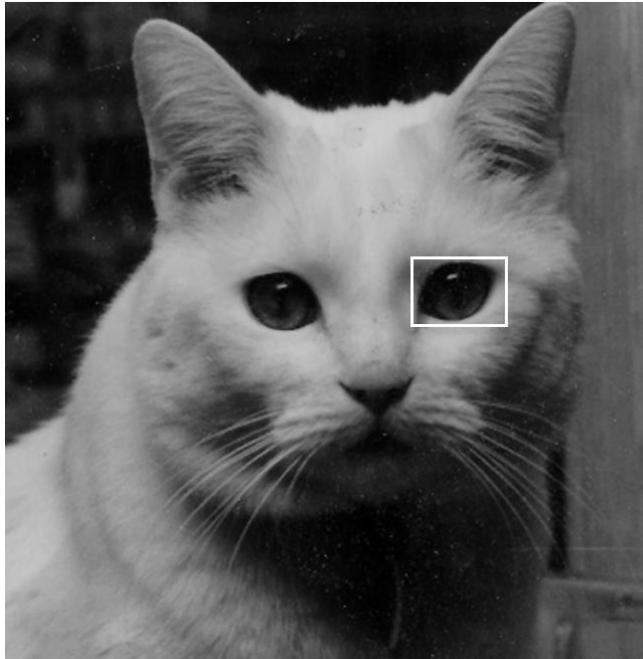


Computer Graphics: 1970 – 1980

- Raster graphics
 - Allows drawing polygons
- First graphics standards
- Workstations and PCs
- WIMP GUI + WYSI**A**WYG
 - Desktop metaphor
 - Selection and direct manipulation

Raster graphics

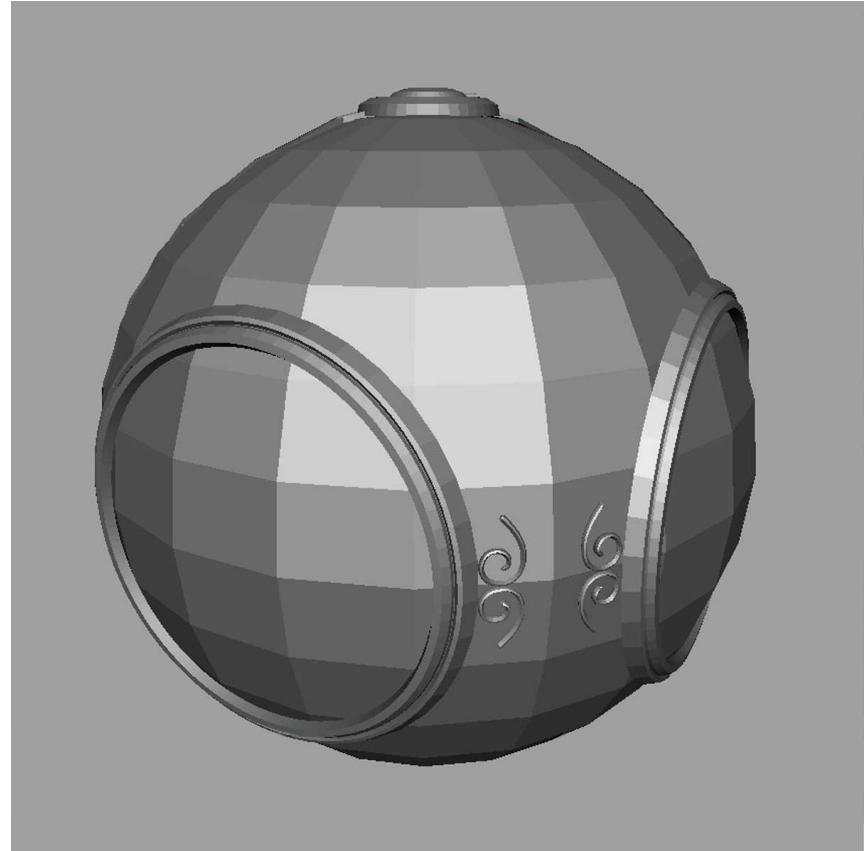
- Image produced as an array (the **raster**) of picture elements (**pixels**) in the **frame buffer**



[Angel]

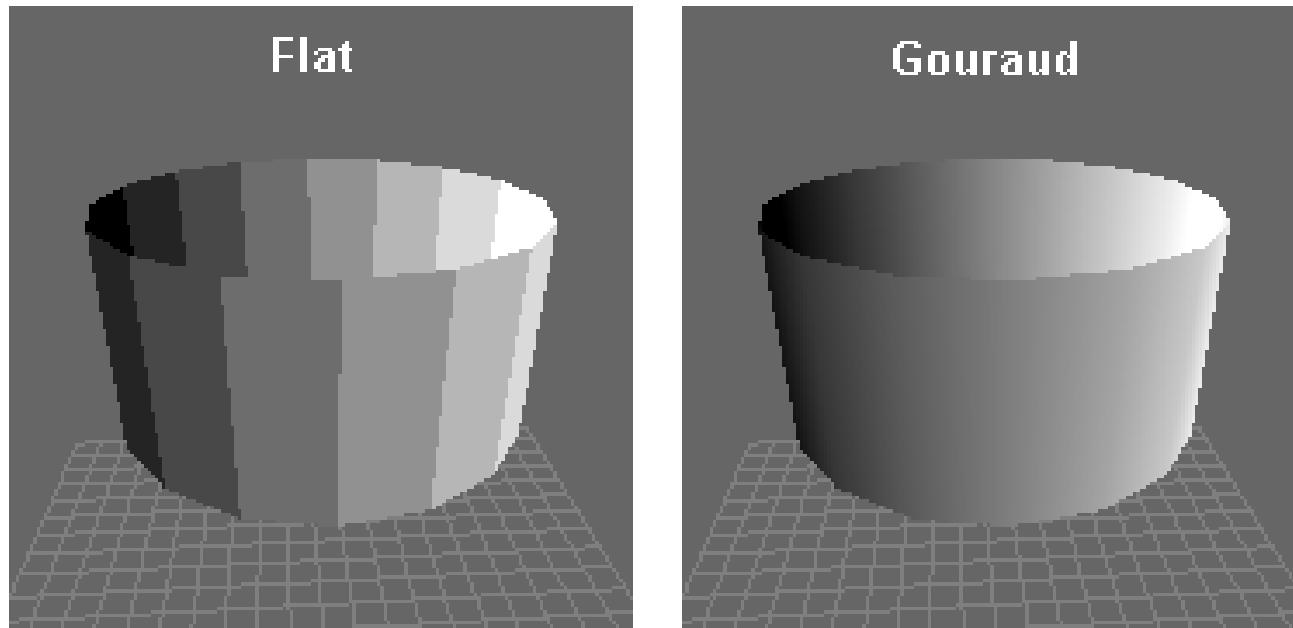
Raster graphics

- Drawing polygons
- Illumination models
- Shading methods



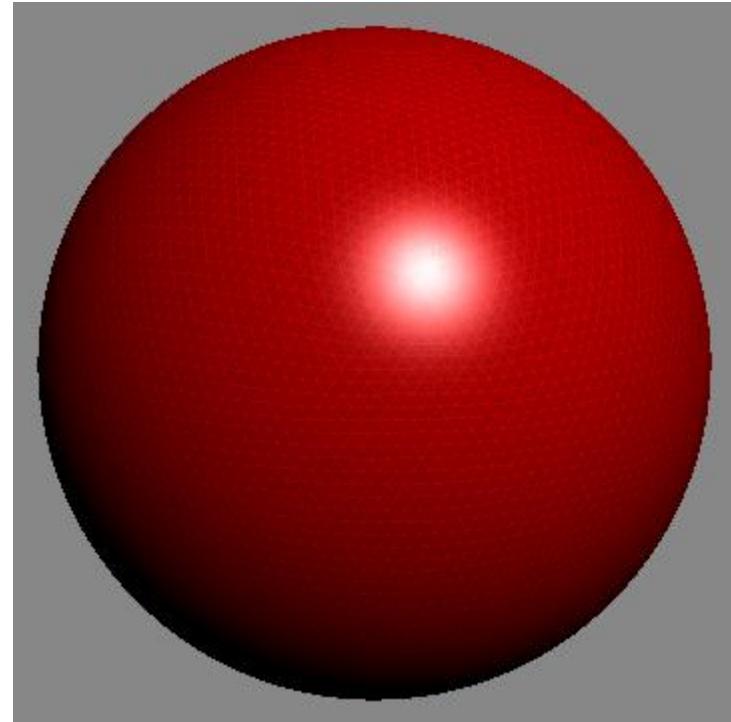
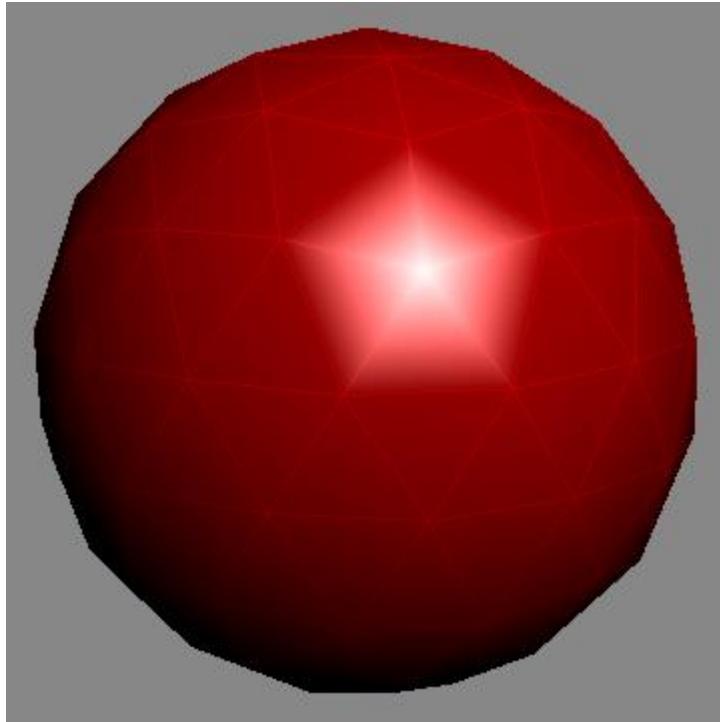
[Angel]

Gouraud shading – 1971



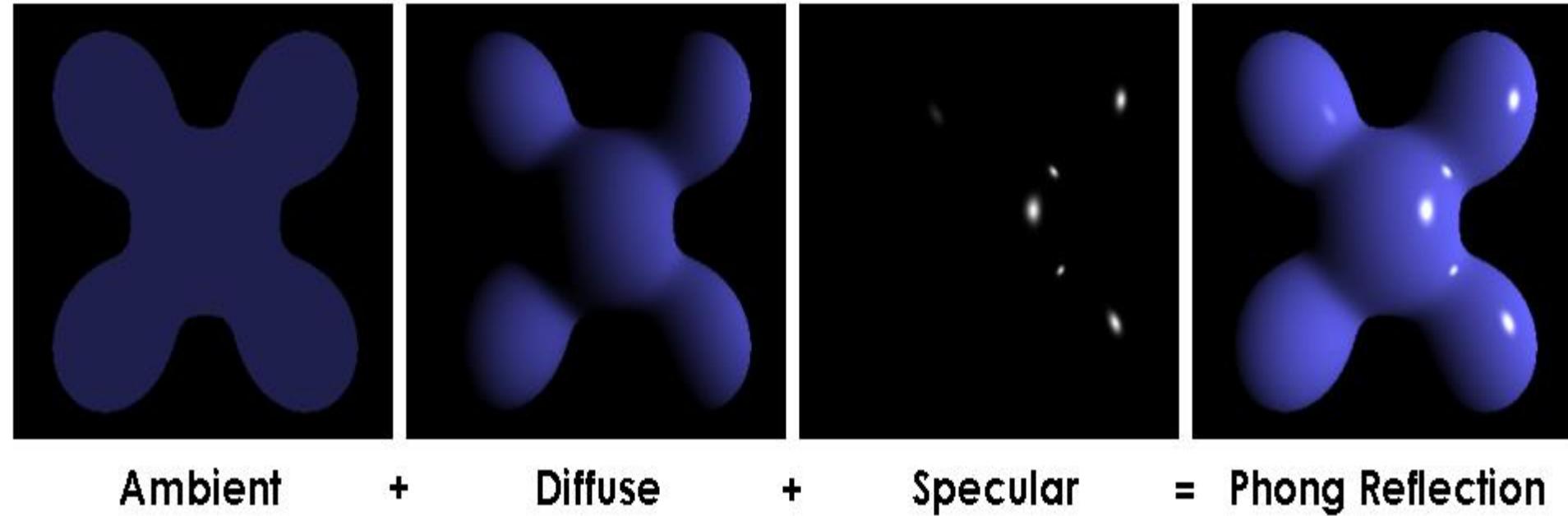
[Wikipedia]

Gouraud shading



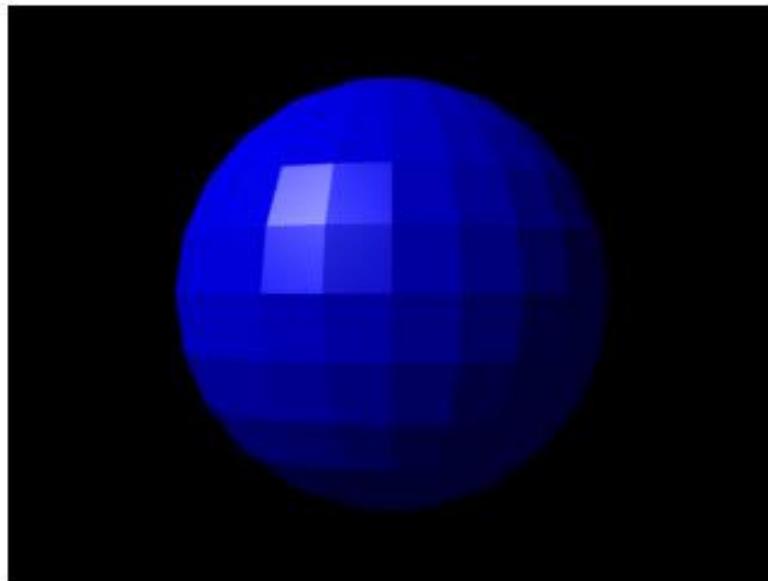
[Wikipedia]

Phong reflection model – 1973

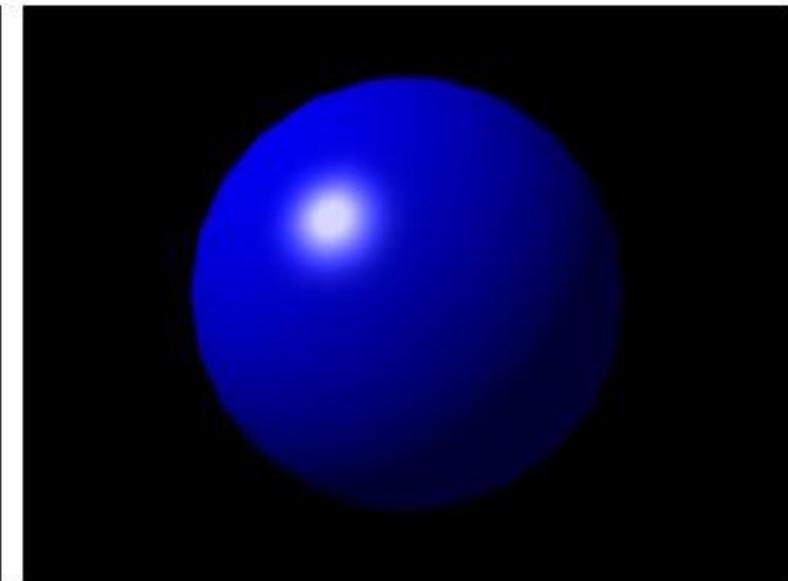


[Wikipedia]

Phong shading – 1973



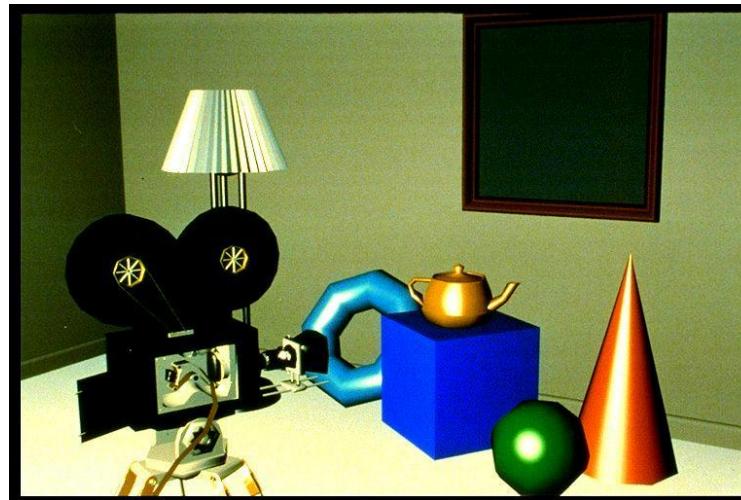
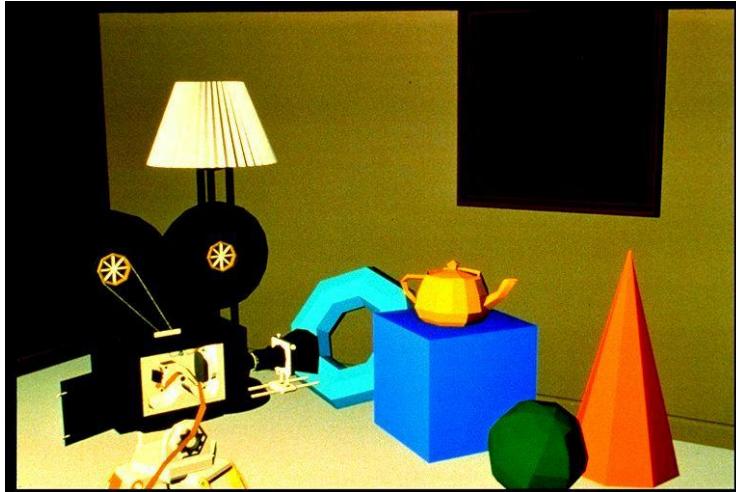
FLAT SHADING



PHONG SHADING

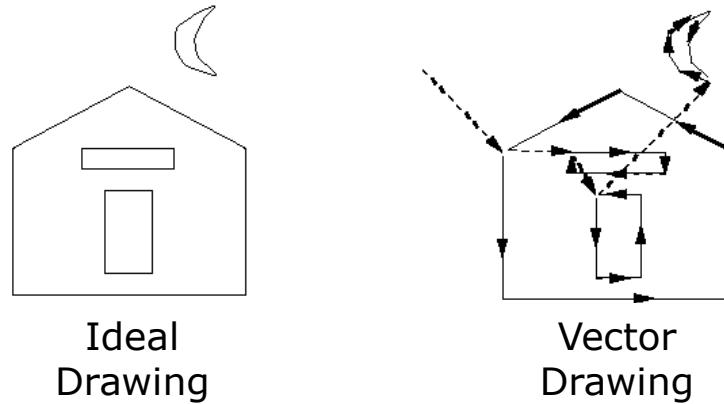
[Wikipedia]

Can you spot the differences ?



Vector graphics vs Raster graphics

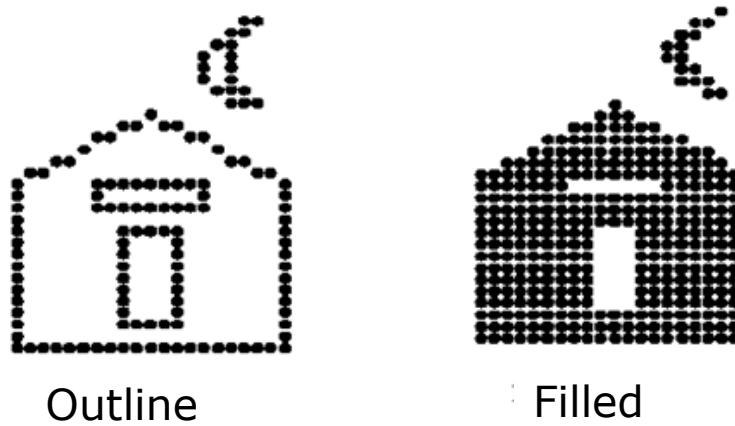
- **Vector graphics** is driven by display commands
 - `move(x,y); line(x,y); ...`
 - Survives as **SVG** – Scalable Vector Graphics



[van Dam]

Vector graphics vs Raster graphics

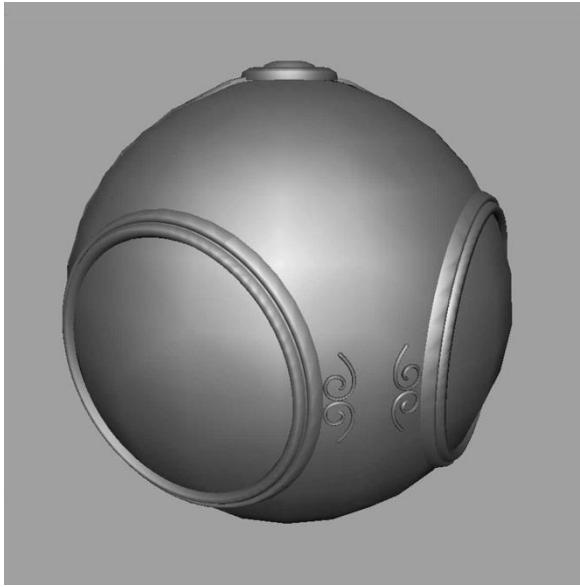
- **Raster graphics** is used in TV displays and laser printers
 - Lowest level of representation
 - No semantics
 - BUT **aliasing errors**



[van Dam]

Computer Graphics: 1980 – 1990

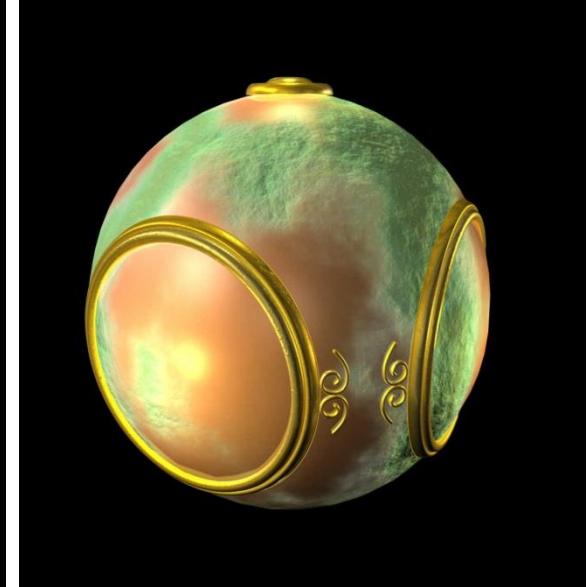
■ The quest for **realism**



Smooth shading



Environment mapping

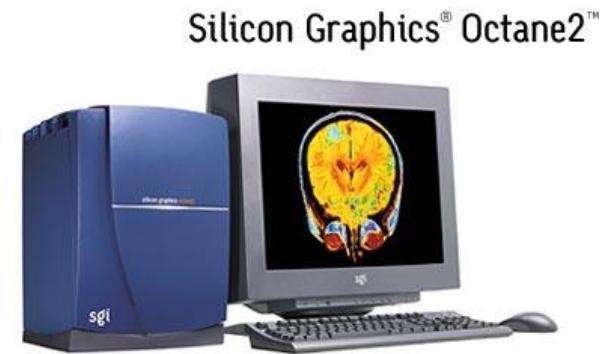


Bump mapping

[Angel]

Computer Graphics: 1980 – 1990

- Special purpose **hardware**
 - Graphics workstations
- Industry-based **standards**
 - PHIGS
 - RenderMan
- Human-Computer Interaction



Silicon Graphics® Octane2™

Graphics workstations such as these have been replaced with commodity hardware (CPU + GPU),

[van Dam]

Computer Graphics: 1990 – 2000

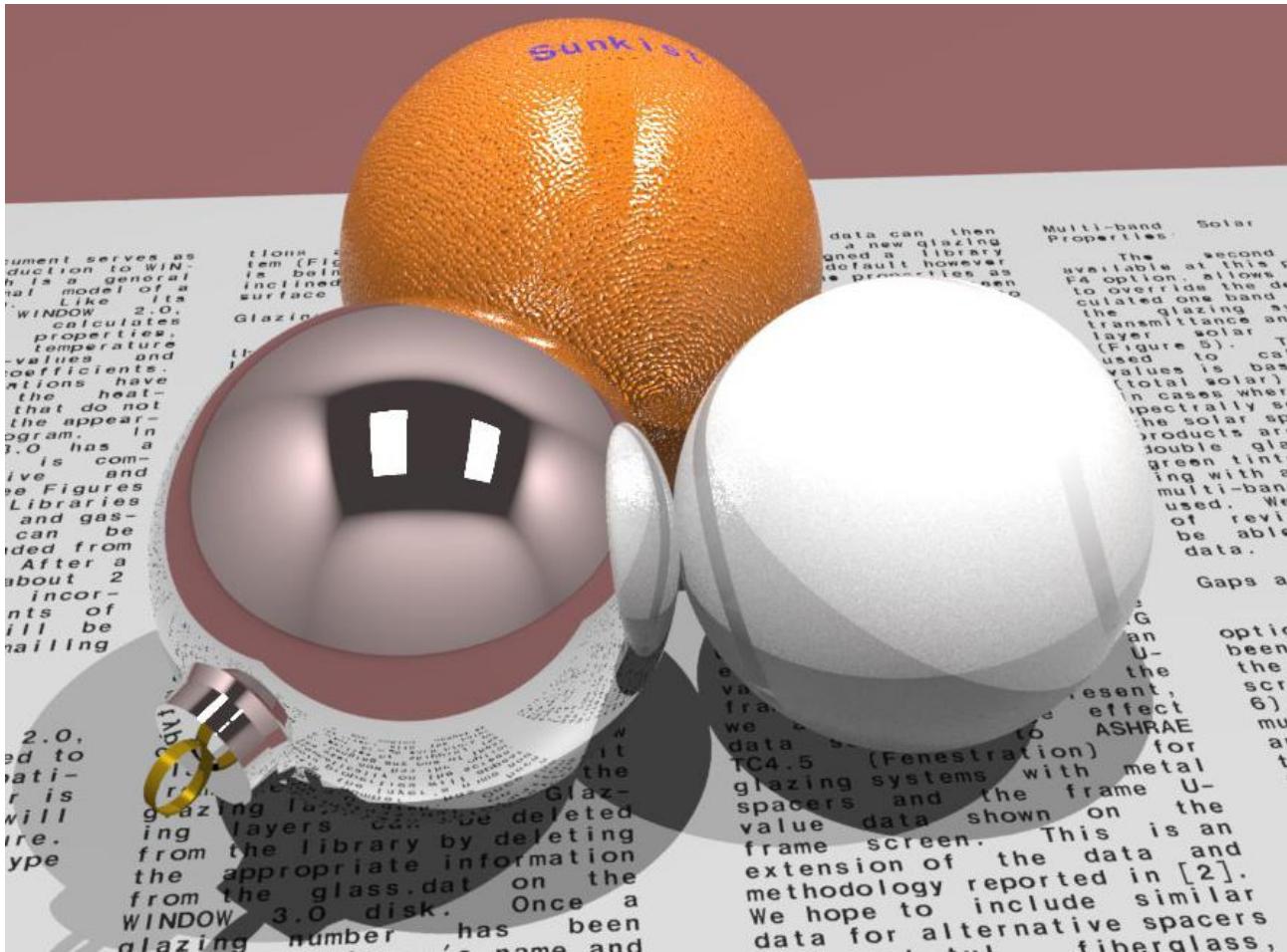
- OpenGL API
- First successful computer-generated feature-length animation film: Toy Story
- New hardware capabilities



Computer Graphics: 2000 – ...

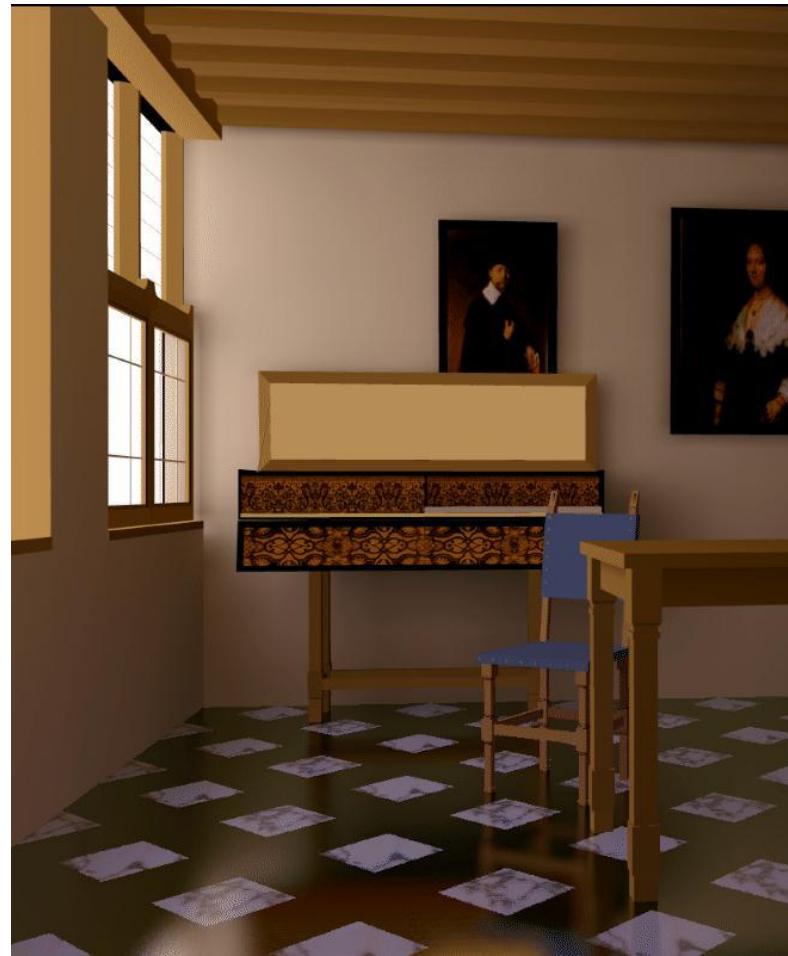
- Photorealism
- Graphics cards for PCs dominate the market
 - Nvidia
 - AMD (ATI)
- Game boxes / players determine the market
- CG is routine in the film industry

Ray-Tracing example



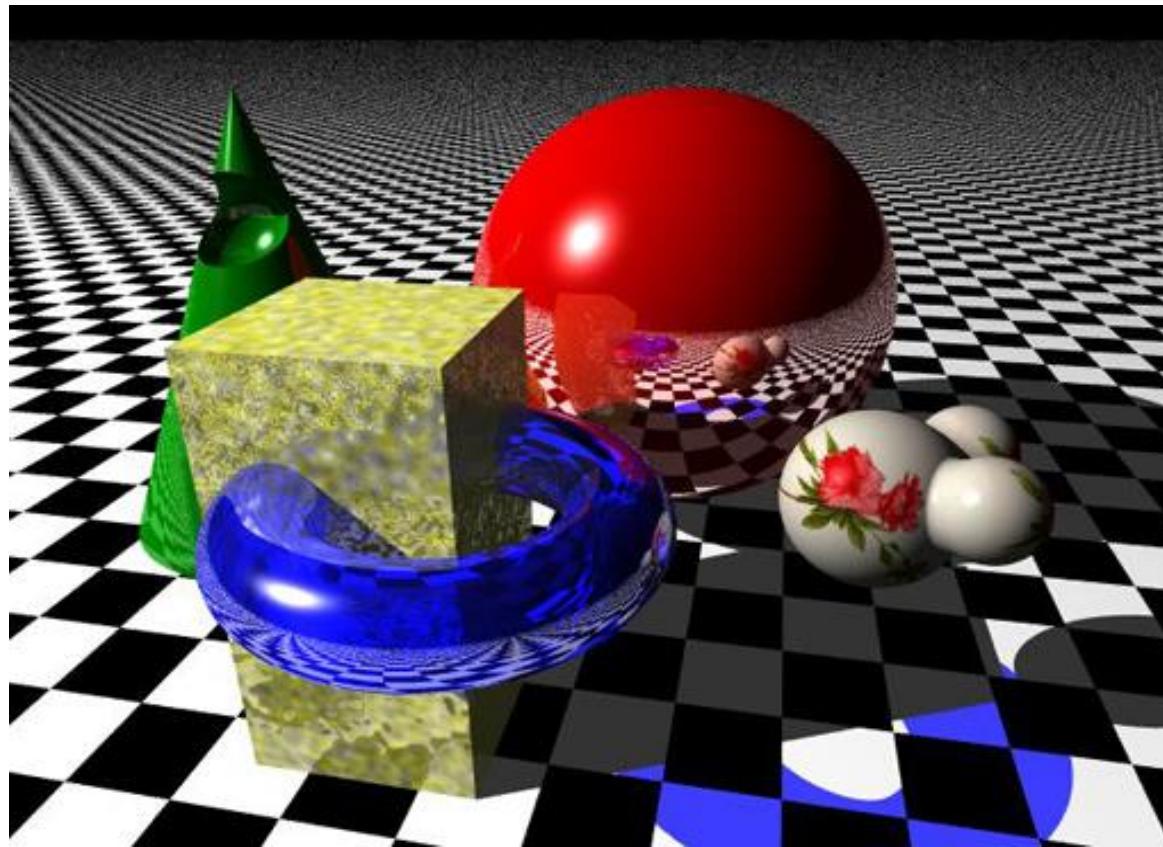
<http://radsite.lbl.gov/radiance/book/img/plate10.jpg>

“Vermeer’s Studio”



Wallace & Cohen, 1987: Radiosity and Ray-Tracing

Another Ray-Tracing example



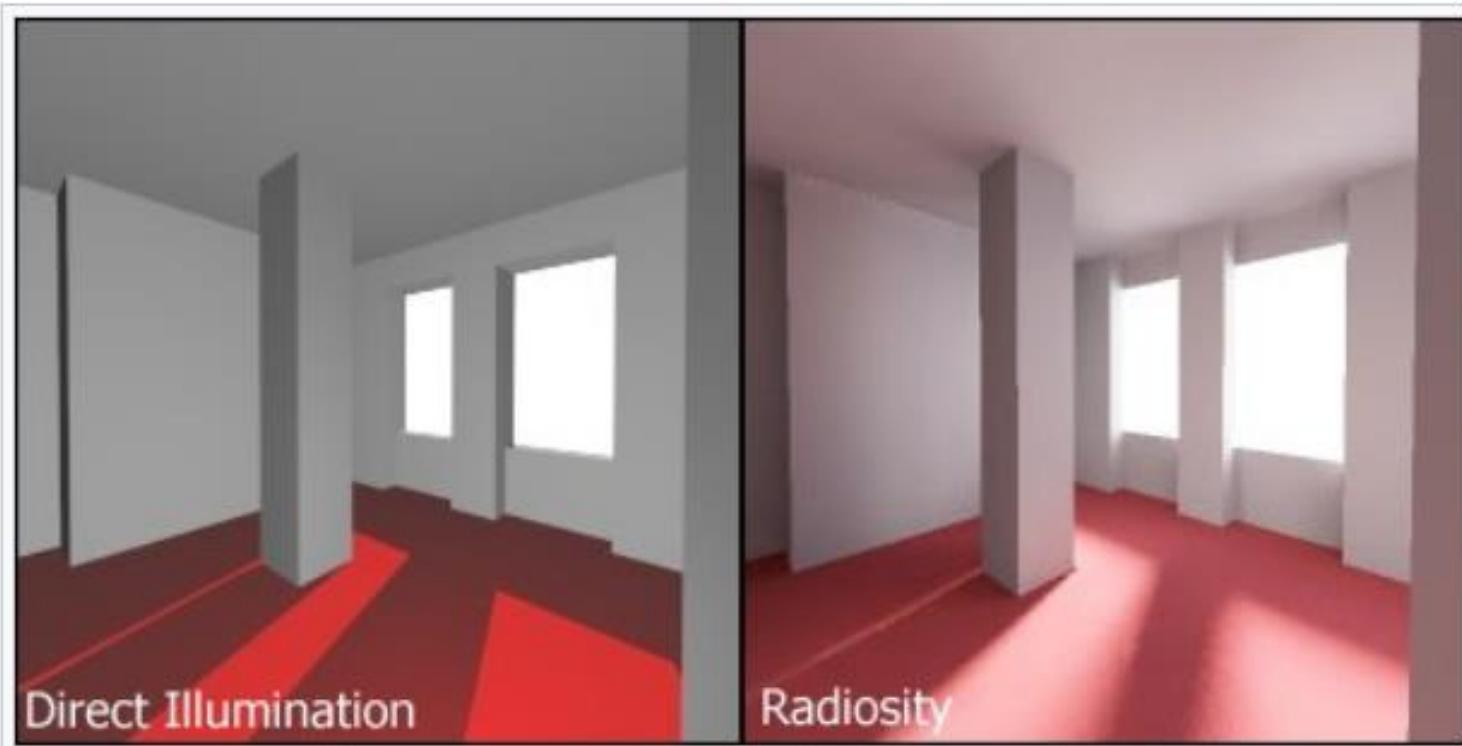
<http://www.tjhsst.edu/~dhyatt/superap/samplex.jpg>

Ray-Tracing + Radiosity



[Wikipedia]

Radiosity



Difference between standard direct illumination without shadow umbra, and radiosity with shadow umbra

[Wikipedia]

Radiosity



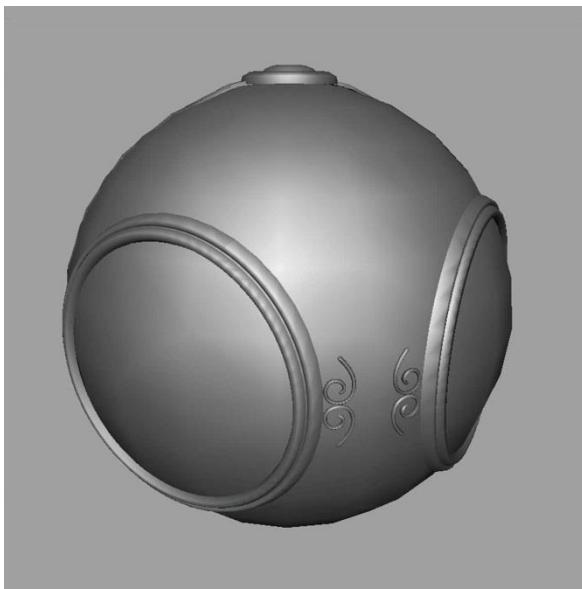
Without radiosity



With radiosity

[Burdea]

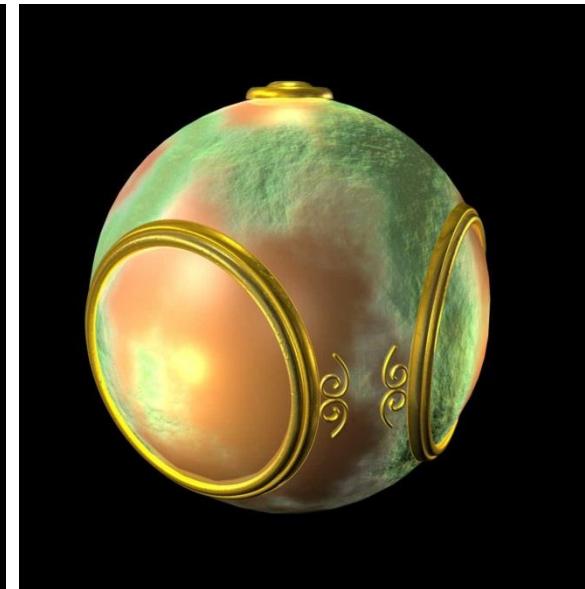
Texture mapping



Smooth shading



Environment mapping



Bump mapping

[Angel]

Textures – Simulating Ray-Tracing



[<http://www.okino.com>]

- Increased realism !!
 - 11 light sources + 25 texture maps

TRENDS & TECHNOLOGIES

Current Panorama

- Augmented Reality used in Hollywood film making
- Virtual Reality WILL be massive !
 - Crash test dummies
 - New car design / development
- Pixar is here !
 - RenderMan is free !!

Current Panorama

- Science and Maths enabling and profiting from advances
 - Modeling / Simulation / Animation
- Gamification
 - Unreal Engine / Unity / Cryengine

Enabling technologies for modern CG

■ Graphics subsystems

- Offload processing from CPU to GPU, for doing graphics operations much quicker

■ Hardware constant “revolution”

- Moore's Law
- Multi-core 64-bit CPUs
- Advances in commodity GPUs every 6 months vs. some years for general purpose CPUs

Enabling technologies for modern CG

- Software improvements
- Algorithms and data structures
 - Modeling of materials
 - Rendering of natural phenomena
 - “Acceleration” data structures for rendering
- Parallelization
 - GPUs

Enabling technologies for modern CG

■ Input devices

- Mouse / tablet & stylus / multi-touch
- Force feedback / game controllers
- Scanners / digitizers / digital cameras
- Body as interaction device



Xbox Kinect



Leap Motion

[Andy van Dam]



Nimble UX

Enabling technologies for modern CG

■ Many **form factors**

- Laptops / desktops
- Smartphones / tablets
- Smartwatches
- HMDs
- Augmented Reality
- Virtual Reality



Microsoft's first Surface



Android Phones



Tablets



Apple Watch



Android Wear



[Andy van Dam]

Microsoft Hololens



Vive

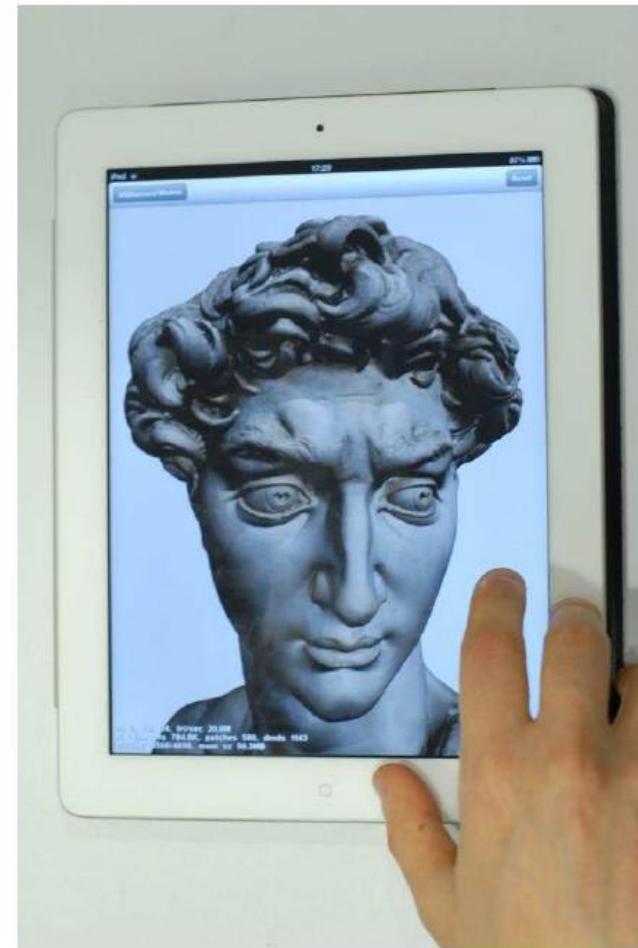


Oculus Rift



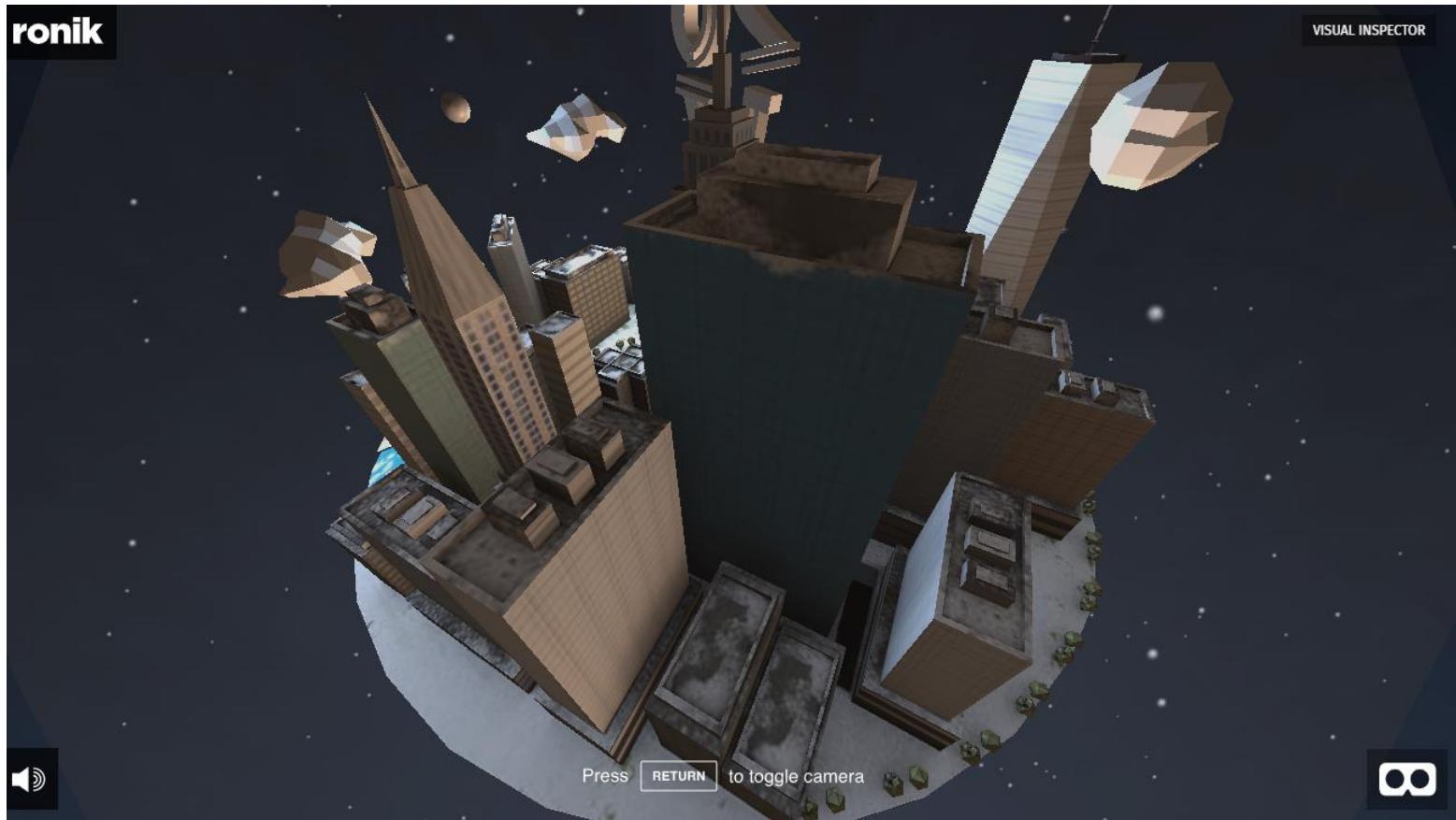
Google Cardboard

Mobile Graphics



[SIGGRAPH Asia 2017 Course Notes]

Web-based VR



[<https://aframe.io/examples/showcase/snowglobe/>]

Web-based CG and VR

- PlayCanvas - The Web-first game engine
- WebXR
- Sketchfab

INTERACTIVE VS BATCH CG

CG Main Tasks

■ Modeling

- Construct individual models / objects
- Assemble them into a 2D or 3D scene

■ Animation

- Static vs. dynamic scenes
- Movement and / or deformation

■ Rendering

- Generate final images
- Where is the observer?
- How is he / she looking at the scene?

Interactive Computer Graphics

- User controls **content**, **structure**, and **appearance** of objects and their displayed images, via rapid **visual feedback**
 - Also called **real-time** computer graphics or, in certain contexts, real-time **rendering**
- Remember **Sutherland's Sketchpad** (1963)
 - Monitor + light pen + function-key panels
 - Bimanual operation

Interactive CG – Basic components

- **Input**
 - Mouse / stylus / multi-touch / in-air fingers / ...
- **Processing** and storing of the underlying models
- **Display / Output**
 - Screen / paper printer / 3D printer / video / ...

Batch Computer Graphics

- Non-interactive, **off-line** rendering
- Final production-quality video and film
 - Animation / Special effects – FX
- Rendering a single frame of *The Good Dinosaur* (a 24 fps movie) averaged **48 hours** on a **30,000-core render farm!**
 - See statistics at [fxguide article](#)

Batch Computer Graphics



Still from *The Good Dinosaur*

[Andy van Dam]



Pixar's Render Farm

MAIN TASKS

CG Main Tasks

■ Modeling

- Construct individual models / objects
- Assemble them into a 2D or 3D scene

■ Animation

- Static vs. dynamic scenes
- Movement and / or deformation

■ Rendering

- Generate final images
- Where is the observer?
- How is he / she looking at the scene?

Modeling vs Rendering

■ Modeling

- Create models
- Apply materials to models
- Place models around scene
- Place lights in the scene
- Place the camera

[YouTube Demo](#)

■ Rendering

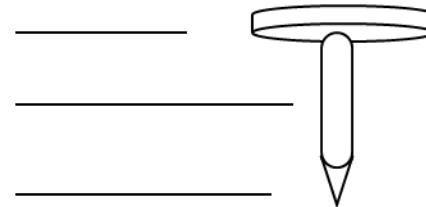
- Take picture with the camera

[van Dam]

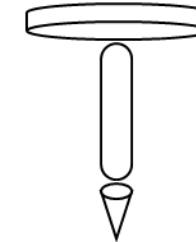
Decomposition of a geometric model

- Hierarchy of geometrical components
- Reduction to primitives
 - Spheres, cubes, etc.
- Simple vs not-so-simple elements

Head
Shaft
Point



composition

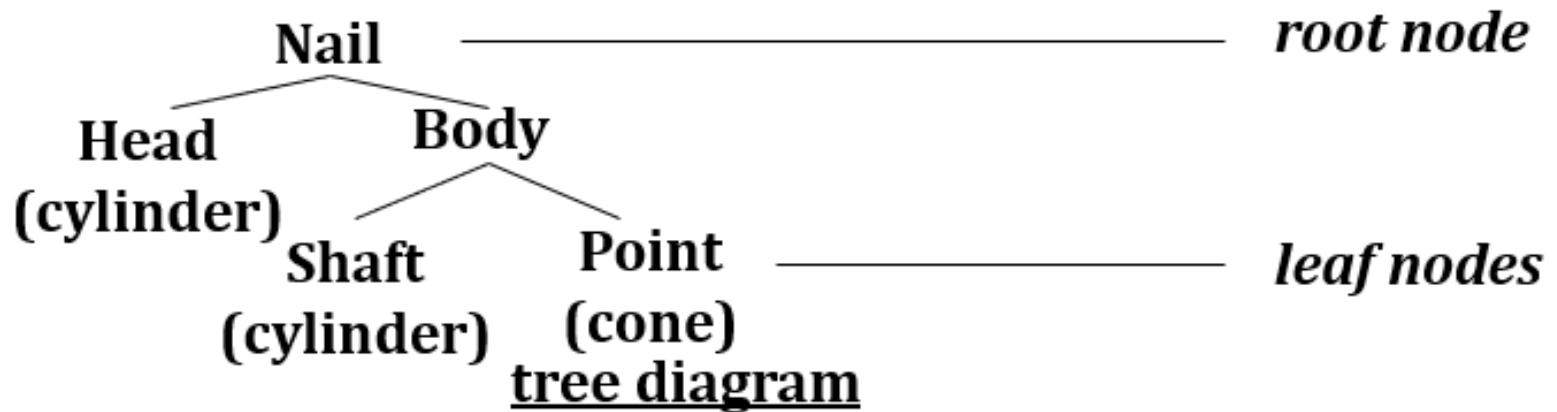


decomposition

[van Dam]

Hierarchical representation

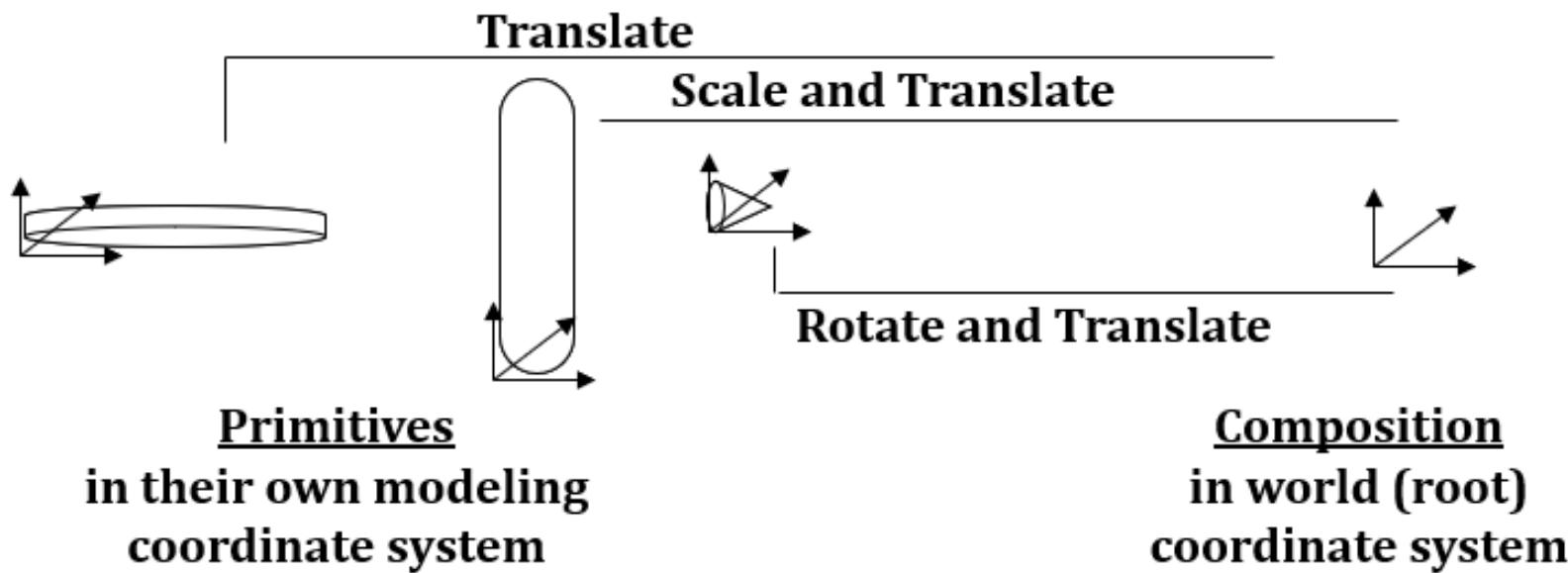
- Decomposition into collections of primitive shapes
 - Tree diagram
- **Scene-graph** : data structure to be rendered



[van Dam]

Composition of a geometric model

- Assemble primitives to create final object
 - Apply affine transformations

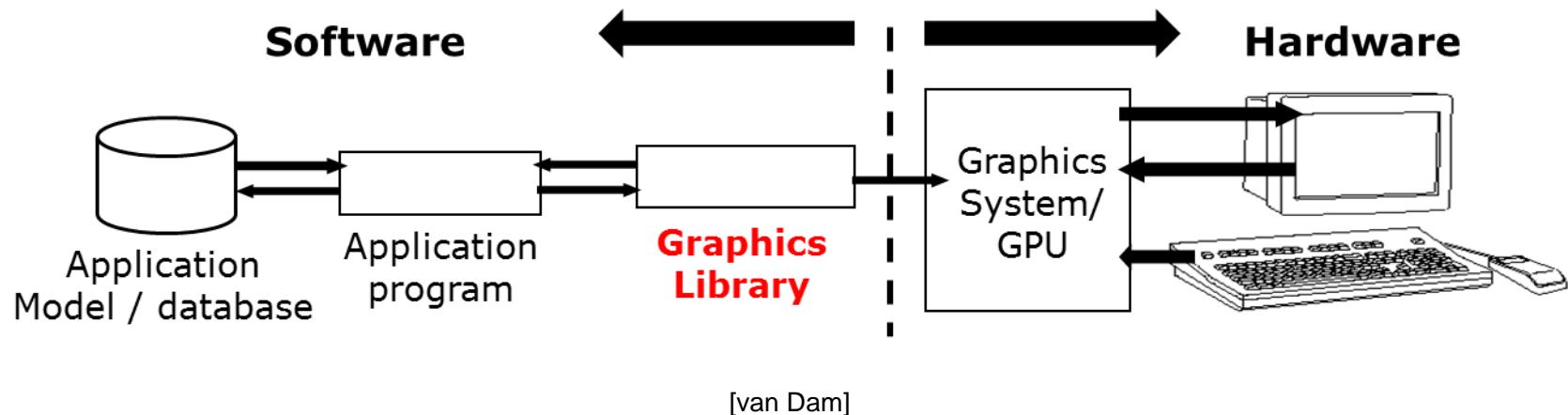


[van Dam]

GRAPHICS APIs / LIBRARIES

Interactive Computer Graphics

- Graphics library / package is **intermediary** between application and display hardware
- Application program **maps / renders** objects / models to images by calling on the **graphics library**
- User **interaction** allows image and / or model modification



Graphics Libraries / APIs

- OpenGL, RenderMan, DirectX, Windows Presentation Foundation (WPF), HTML5 + **WebGL**, **three.js**, Vulkan,
...
- **Primitives** : characters, points, lines, triangles, ...
- **Attributes** : color, line /polygon style, ...
- **Transformations** : rotation, scaling, ...
- **Light sources**
- **Viewing**
- ...



API contents

- Functions for specifying / instantiating
 - Geometric primitives
 - Viewer / Camera
 - Light sources
 - Materials
 - ...
- Functions for simple user interaction
 - Input from devices: mouse, keyboard, etc.

Geometric Primitives

■ Simple primitives

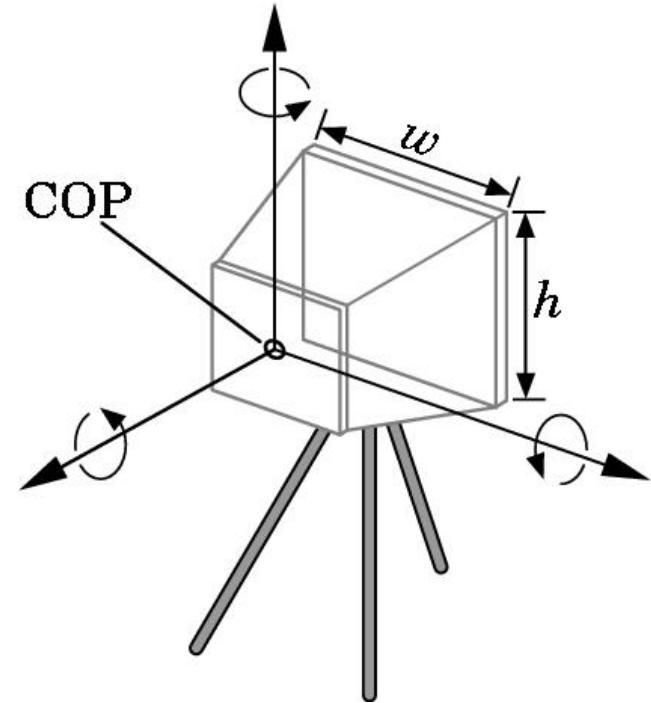
- Points
- Line segments
- Polygons

■ Geometric primitives

- Parametric curves / surfaces
- Cubes, spheres, cylinders, etc.

Camera specification

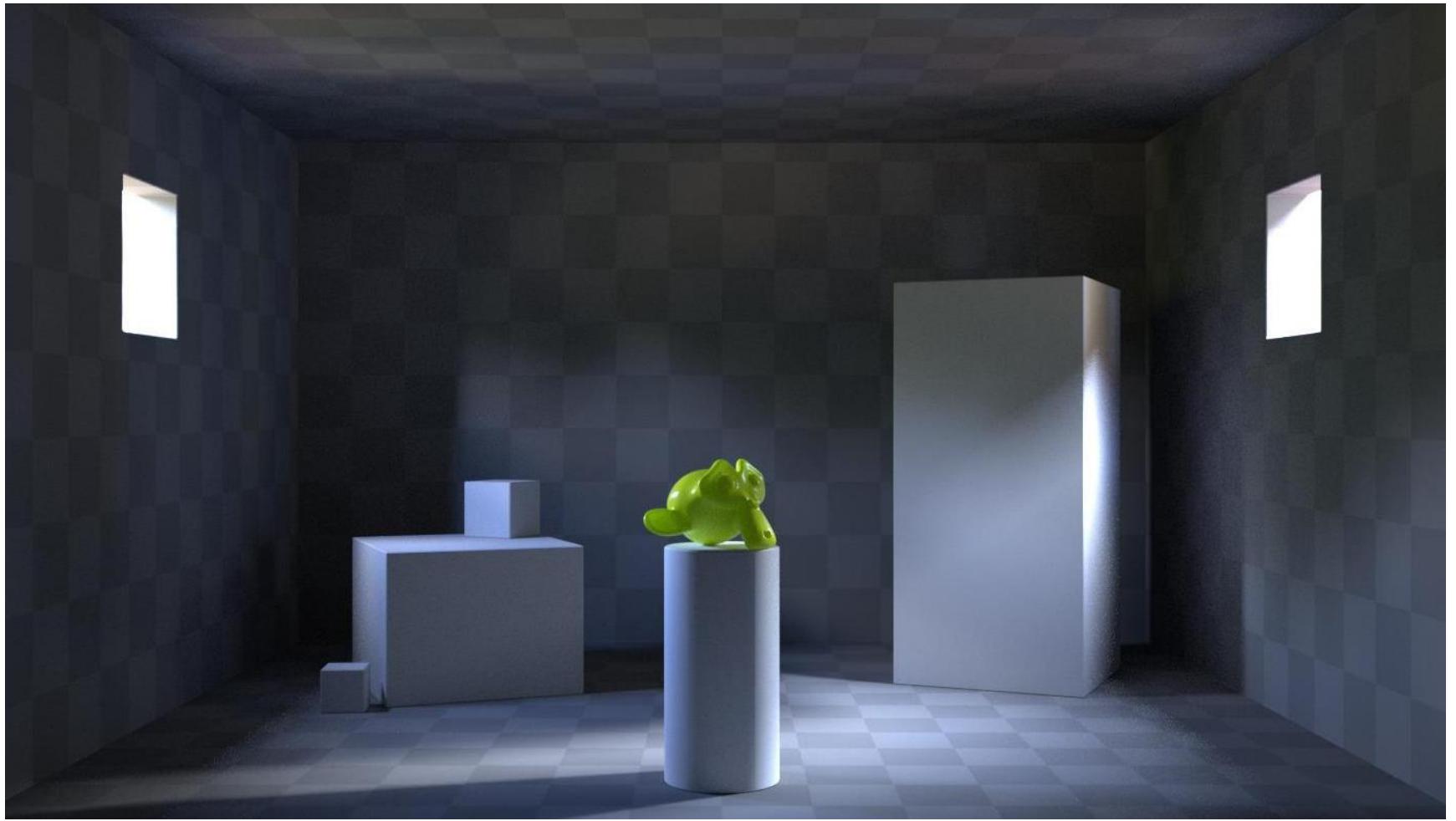
- Six degrees of freedom
 - Position of lens center
- Lens
- Film size
- Orientation of film plane



[Angel]

Lights and materials

- Types of light sources
 - Point vs distributed light sources
 - Spotlights
 - Near and far sources
 - Color properties
- Material properties
 - Absorption: color properties
 - Scattering: diffuse and specular
 - Transparency



<https://blenderartists.org/t/what-if-i-want-a-low-light-in-door-scene/685160>

OpenGL



- Multi-platform API for rendering 2D and 3D computer graphics
- Interaction with the GPU to achieve hardware-accelerated rendering
- Application areas
 - CAD
 - Virtual reality
 - Scientific and Information Visualization
 - ...

OpenGL



■ OpenGL ES

- Subset for use in embedded systems and portable devices



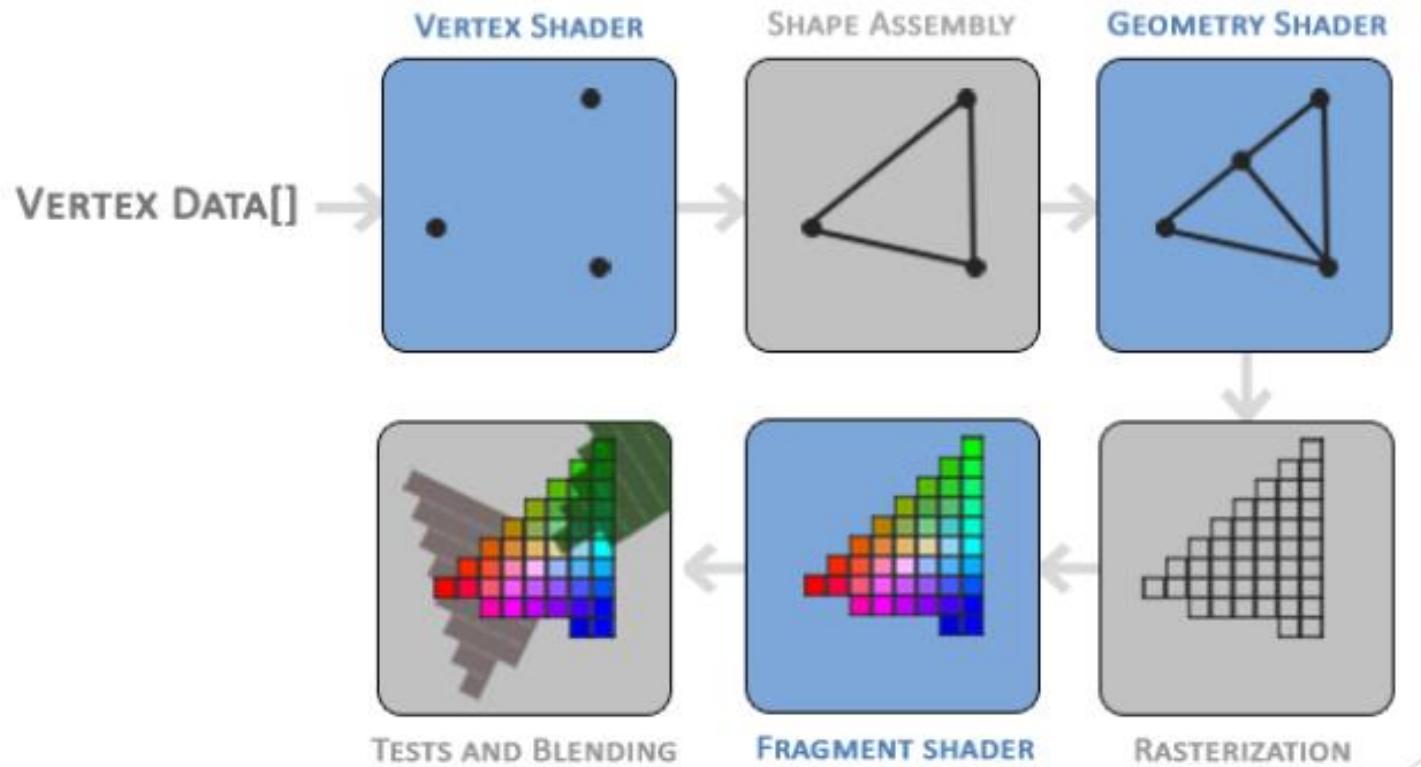
■ WebGL

- JavaScript API based on OpenGL ES 2.0
- Rendering interactive 2D and 3D graphics on any compatible browser, without the use of plug-ins

GPUs

- Large collection of **highly parallel high-speed arithmetic units**; several thousand cores !
- GPUs run simple programs (“shaders”)
 - Take in **vertices** and **other data**
 - Output a **color value** for an **individual pixel**
- **GLSL**, (O)GL Shader Language, is a C-like language; controls arithmetic pipelines

Shaders



learnopengl.com

Three.js

- Create and display **3D CG** in **Web-browsers**
 - Cross-browser **Javascript API**
 - No need for browser plug-ins !
- No need to develop stand-alone apps !
- Uses **WebGL** !
- threejs.org

Three.js

three.js r97

featured projects

[documentation](#)
[examples](#)

[download](#)

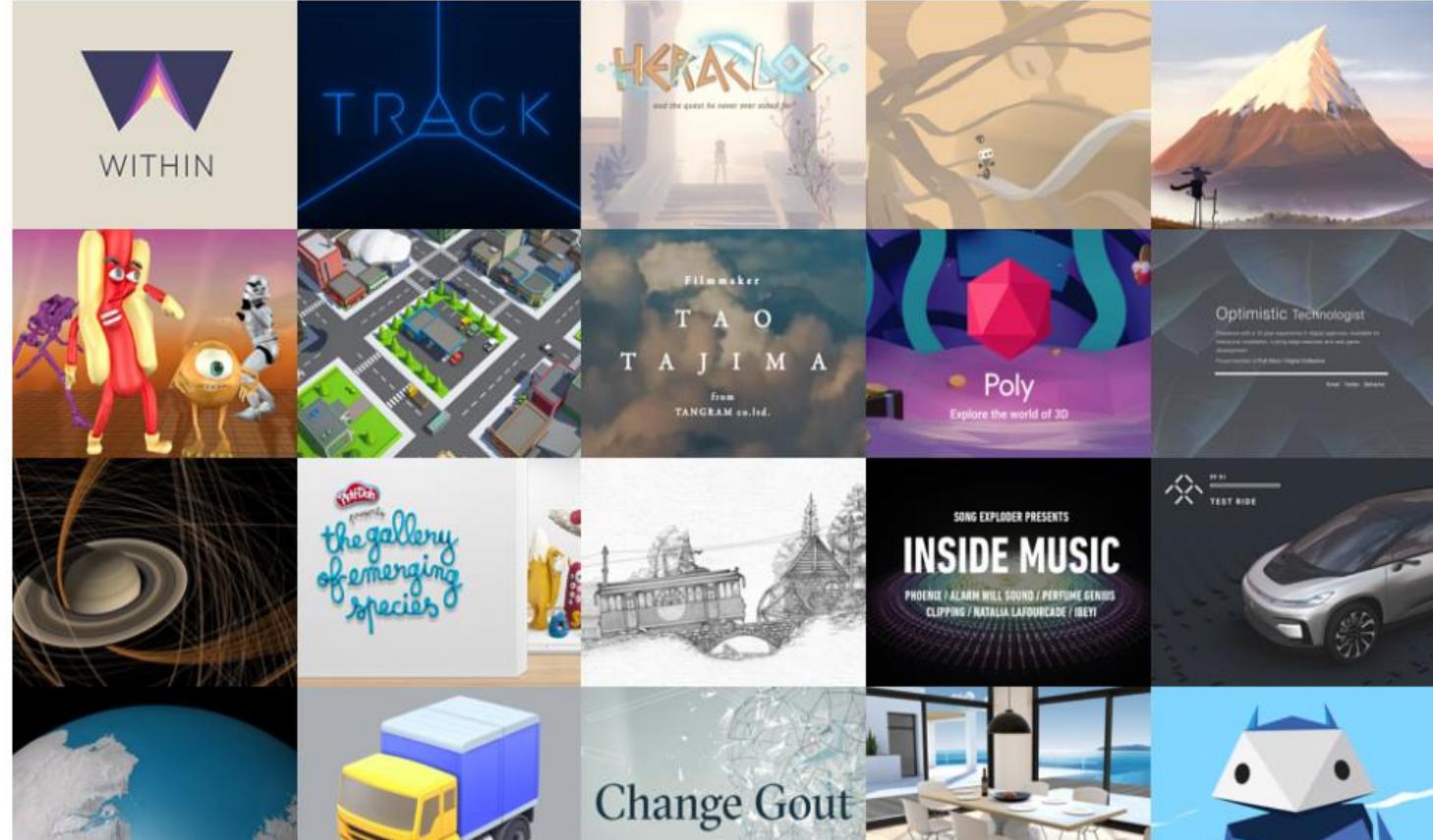
[source code](#)
[questions](#)
[forum](#)
[irc](#)
[slack](#)
[google+](#)



Interactive
3D Graphics
Taught by Eric Haines



UDACITY



RECAP

In Summary

- Computer graphics involves both **real-time / interactive** and **batch / offline** applications
 - Both equally important, but different use cases
- **Photo-realism** has really advanced !!
 - But it still takes dozens of hours on fastest computers to mimic physical behavior

In Summary

- Hardware evolution from vector to **raster graphics**
 - But we still have **SVG**
- **Geometry-based vs Image-based** graphics
 - Mathematical definition vs pixel manipulation

In Summary

- Complex geometric models typically constructed **hierarchically**
 - Scene-graph data structure
- **Pixels** are discrete samples of continuous functions
 - Causes **artifacts** (“jaggies”/ “aliases”) to appear
 - Need fixing through “anti-aliasing”

In Summary

- **Geometric models** allow representing objects or characters
 - Geometry / Detail / ...
 - Material / Appearance
 - Static vs Animated / Deformable models
- **Scene modeling**
 - Place models / light sources
 - Place the camera

REFERENCES

References

- S. Marschner, P. Shirley, “Fundamentals of Computer Graphics”, 4th ed, A K Peters, 2018
 - Chapter 1
 - <https://learning.oreilly.com/library/view/fundamentals-of-computer/9781482229417/>
- J. F. Hughes, A. van Dam, et al., “Computer Graphics: Principles and Practice”, 3rd ed, Addison-Wesley, 2013
 - Chapter 1
 - <https://learning.oreilly.com/library/view/hughes-computer-graphics-3-e/9780133373721/>

ACKNOWLEDGMENTS

Acknowledgments

- Some ideas and figures have been taken from slides of other CG courses.
- In particular, from the slides made available by Ed Angel, Andy van Dam, and Samuel Silva.
- Thanks!

An Introduction to Three.js

Joaquim Madeira

March 2022

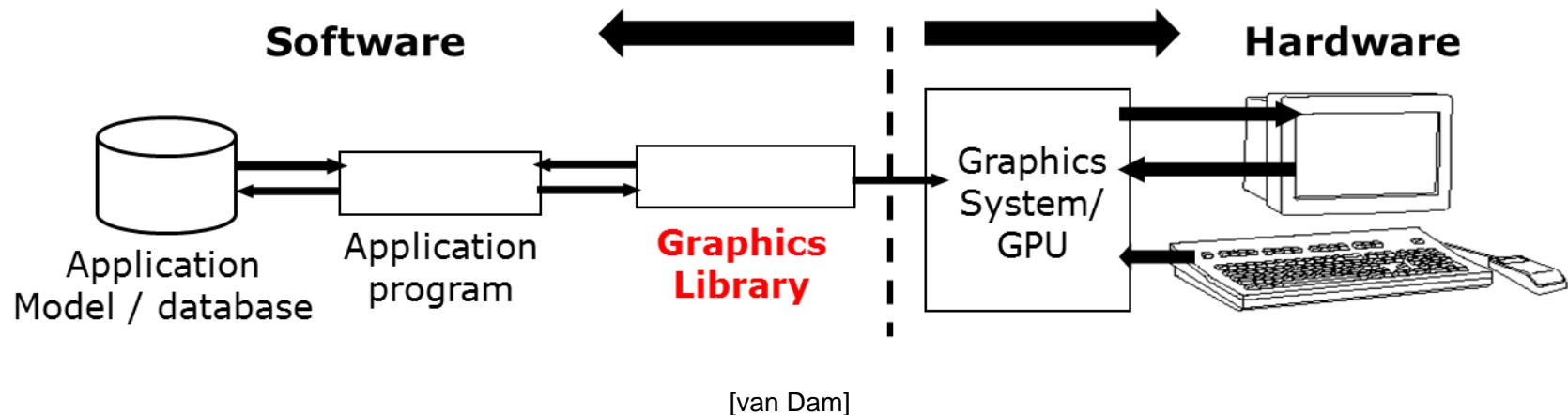
Topics

- Three.js – Main features

RECAP

Interactive Computer Graphics

- Graphics library / package is **intermediary** between application and display hardware
- Application program **maps / renders** objects / models to images by calling on the **graphics library**
- User **interaction** allows image and / or model modification



CG Main Tasks

■ Modeling

- Construct individual **models** / objects
- Assemble them into a 2D or 3D **scene**

■ Animation

- Static vs. dynamic scenes
- Movement and / or deformation

■ Rendering

- Generate final images
- Where is the viewer / camera ?
- How is he / she looking at the scene?

Modeling vs Rendering

■ Modeling

- Create models
- Apply materials to models
- Place models around scene
- Place lights in the scene
- Place the camera

[YouTube Demo](#)

■ Rendering

- Take picture with the camera

[van Dam]

THREE.JS

– MAIN FEATURES

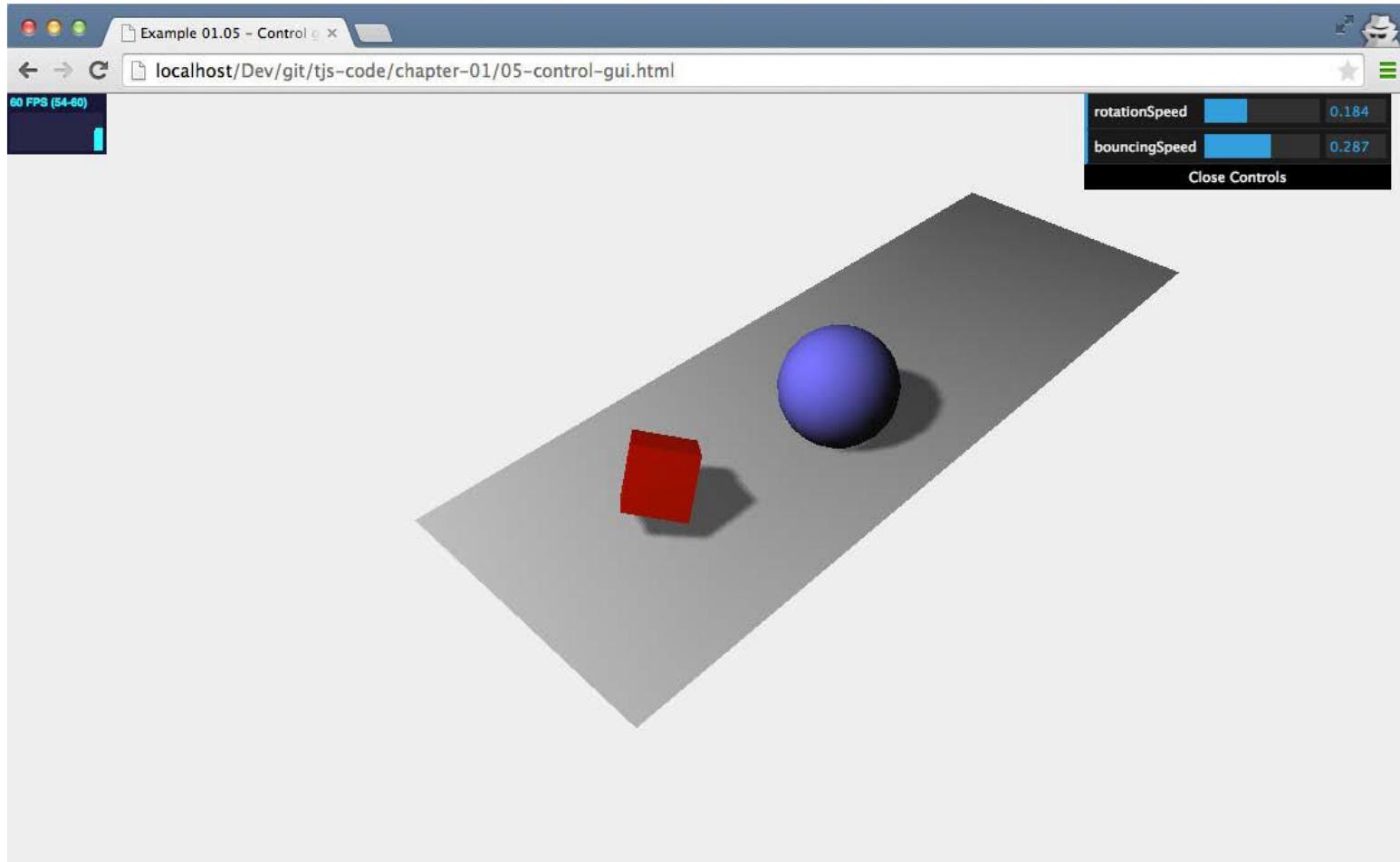
What can we do?

- With a few lines of JavaScript...
- Create anything, from simple 3D models to photorealistic real-time scenes

What can we do?

- Create simple and complex 3D geometries
- Animate and move objects through a 3D scene
- Apply textures and materials to objects
- Make use of different light sources to illuminate the scene

A simple scene

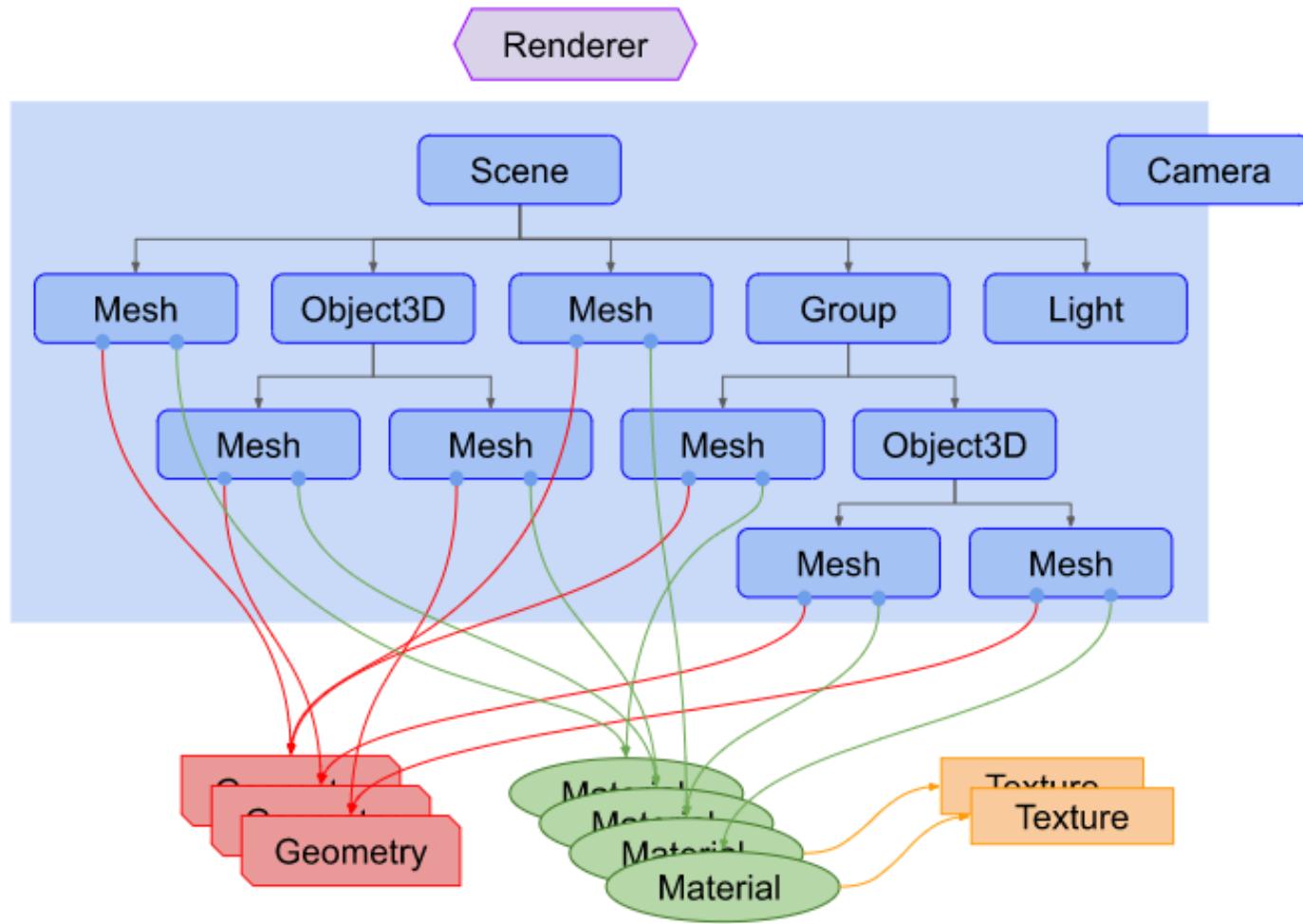


What can we do?

- Load objects from 3D-modeling software
- Add advanced postprocessing effects to a 3D scene
- Work with custom shaders
- Create point clouds

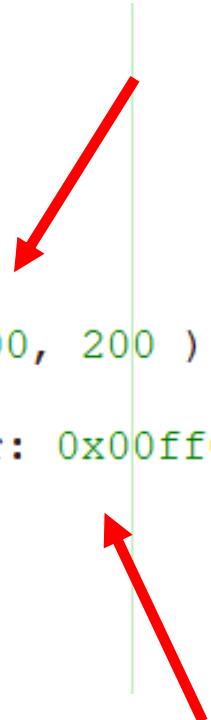
THE BASICS

Three.js – App structure



Three.js – A scene with a cube

```
// The SCENE  
  
scene = new THREE.Scene();  
  
// The MODEL --- A cube is added to the scene  
  
var geometry = new THREE.BoxBufferGeometry( 200, 200, 200 );  
  
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );  
  
model = new THREE.Mesh( geometry, material );  
  
scene.add( model );
```



Three.js – The camera

```
// The CAMERA  
  
// --- Where the viewer is and how he is looking at the scene  
  
camera = new THREE.PerspectiveCamera( 70,  
    window.innerWidth / window.innerHeight, 1, 1000 );  
  
camera.position.z = 400;
```



Three.js – The renderer

```
// The RENDERER --- To display the scene on the Web page  
  
renderer = new THREE.WebGLRenderer( { antialias: true } );  
  
renderer.setPixelRatio( window.devicePixelRatio );  
  
renderer.setSize( window.innerWidth, window.innerHeight );  
  
document.body.appendChild( renderer.domElement );
```



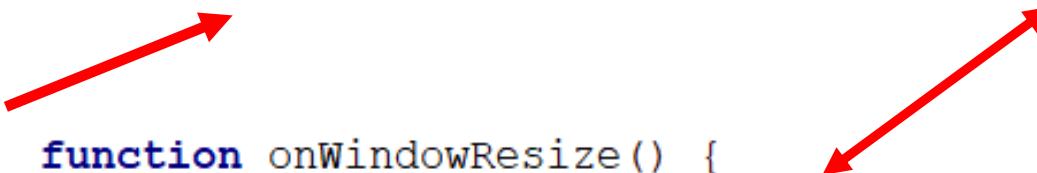
Three.js – Animation

```
function animate() {  
    requestAnimationFrame( animate );  
  
    model.rotation.x += 0.005;  
  
    model.rotation.y += 0.01;  
  
    renderer.render( scene, camera );  
}
```



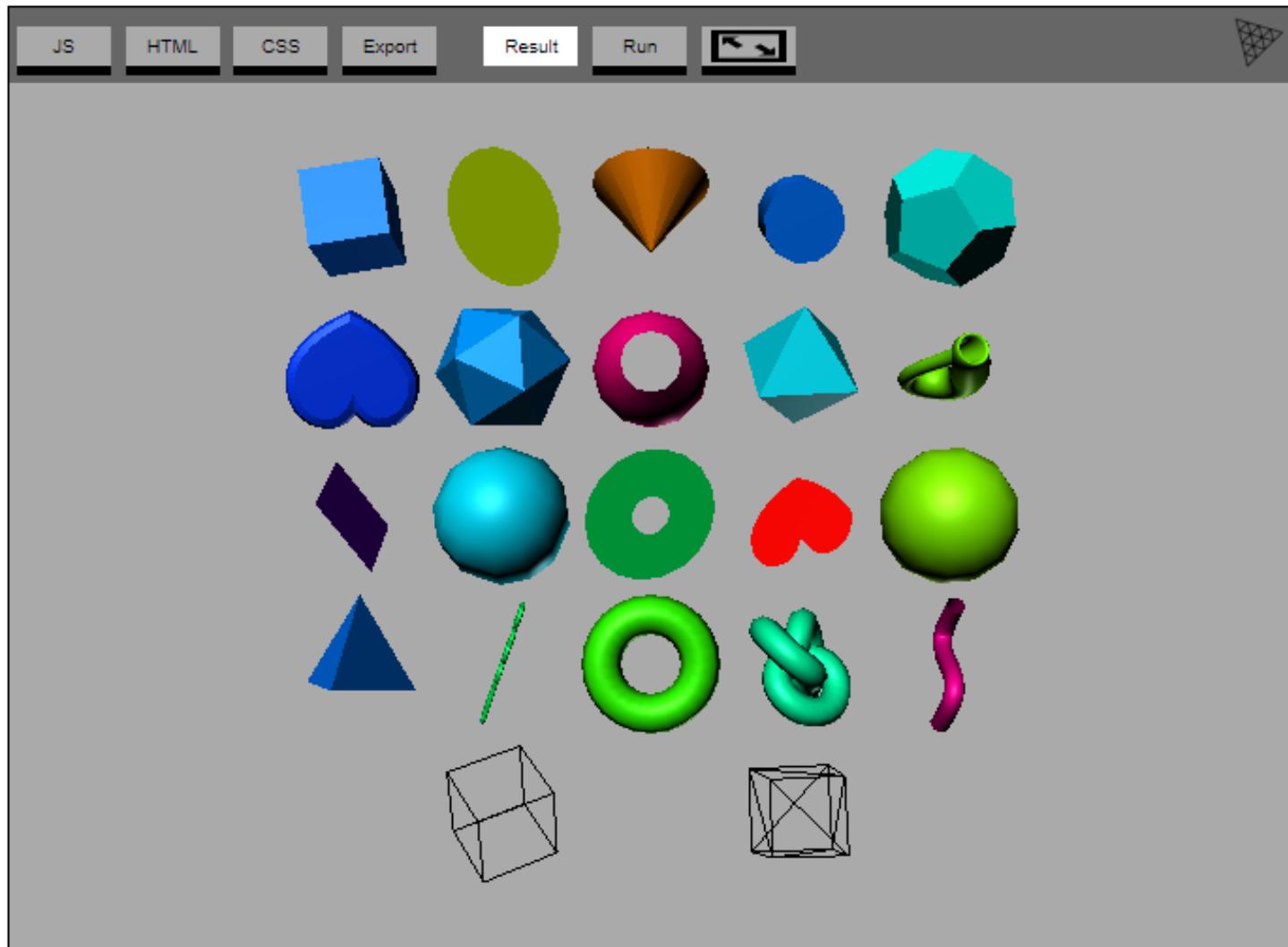
Three.js – Interaction

```
// What to do if resize occurs ?  
  
window.addEventListener( 'resize', onWindowResize, false );  
  
function onWindowResize() {  
    // Adjusting the renderer and camera features  
  
    renderer.setSize( window.innerWidth, window.innerHeight );  
  
    camera.aspect = window.innerWidth / window.innerHeight;  
  
    camera.updateProjectionMatrix();  
}
```



3D GEOMETRIES

Three.js – 3D Geometries



Three.js – 3D Geometries

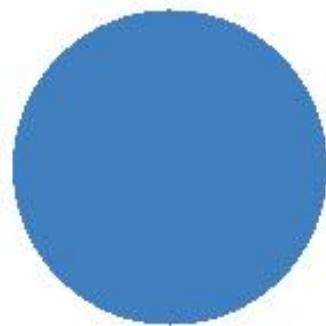
The screenshot shows a portion of the Three.js documentation website. At the top, there is a navigation bar with three tabs: "three.js" (which is highlighted in blue), "docs", and "examples". Below the navigation bar is a search bar with a magnifying glass icon and a language selector set to "en". The main content area displays a list of geometry types under the heading "Geometries". The list includes:

- BoxGeometry
- CircleGeometry
- ConeGeometry
- CylinderGeometry
- DodecahedronGeometry
- EdgesGeometry
- ExtrudeGeometry
- IcosahedronGeometry
- LatheGeometry
- OctahedronGeometry
- PlaneGeometry
- PolyhedronGeometry
- RingGeometry
- ShapeGeometry
- SphereGeometry

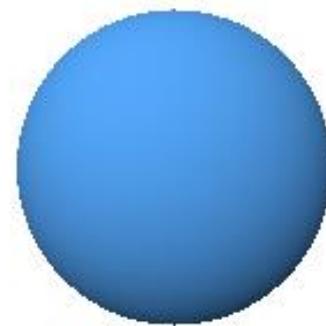
At the bottom of the list, there are three small navigation icons: a left arrow, a right arrow, and a double arrow.

MATERIALS & ILLUMINATION

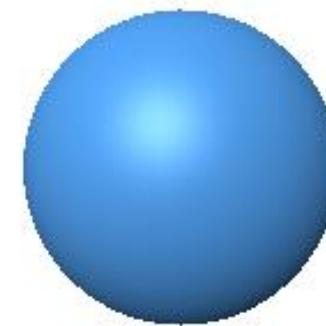
Three.js – Materials & Level-of-Detail



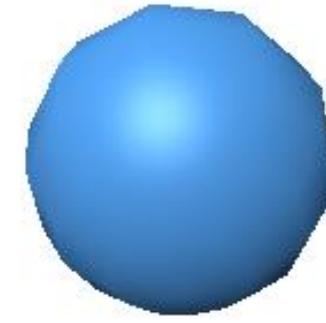
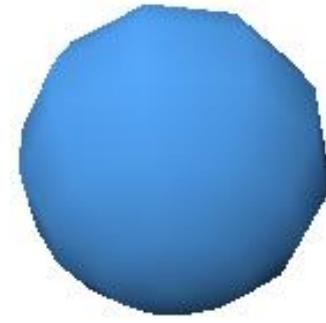
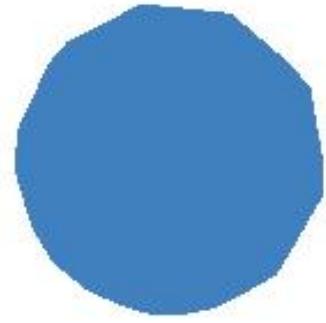
Basic



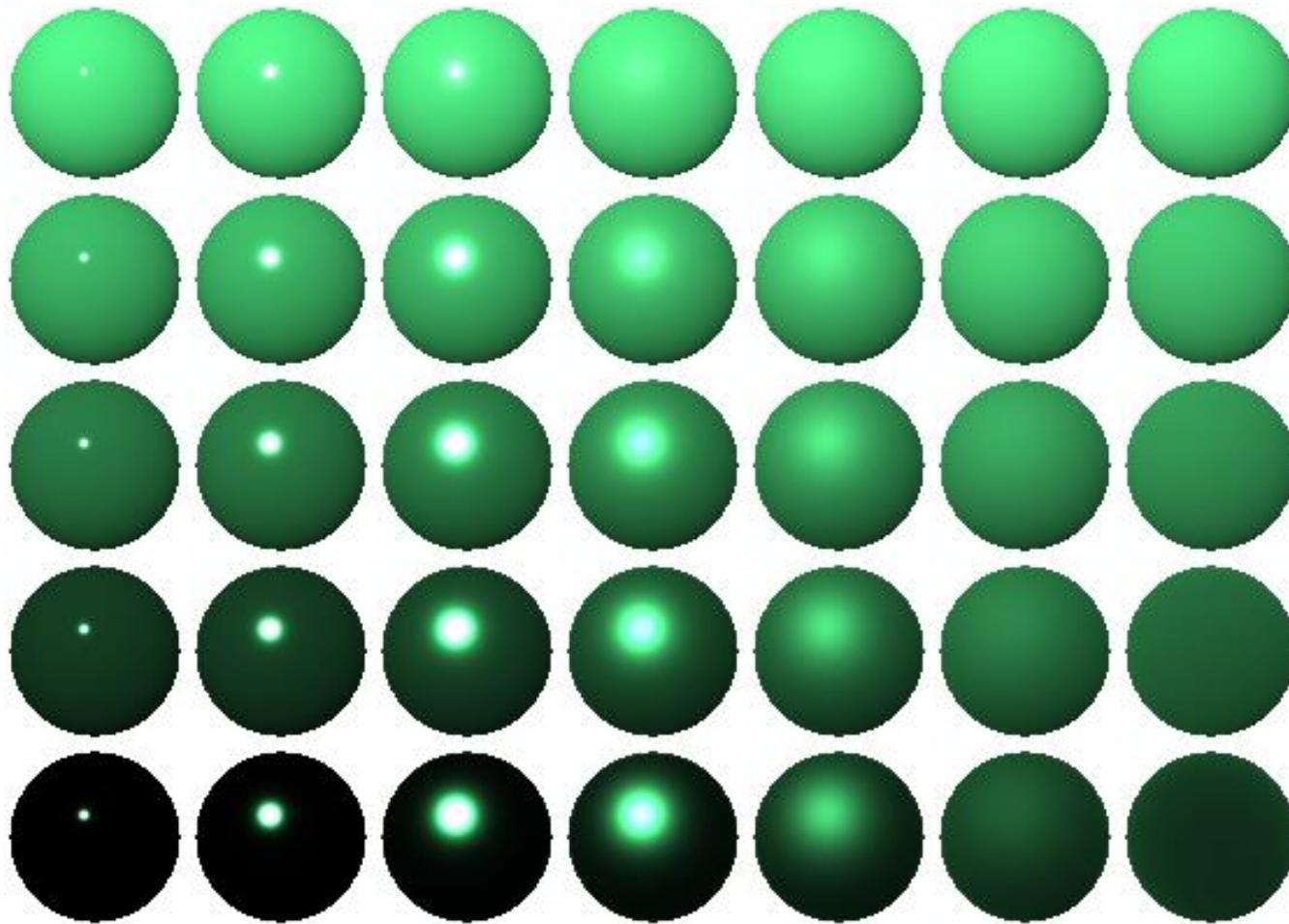
Lambert



Phong



Three.js – Illumination effects



Three.js – Materials

[three.js](#) [docs](#) [examples](#)



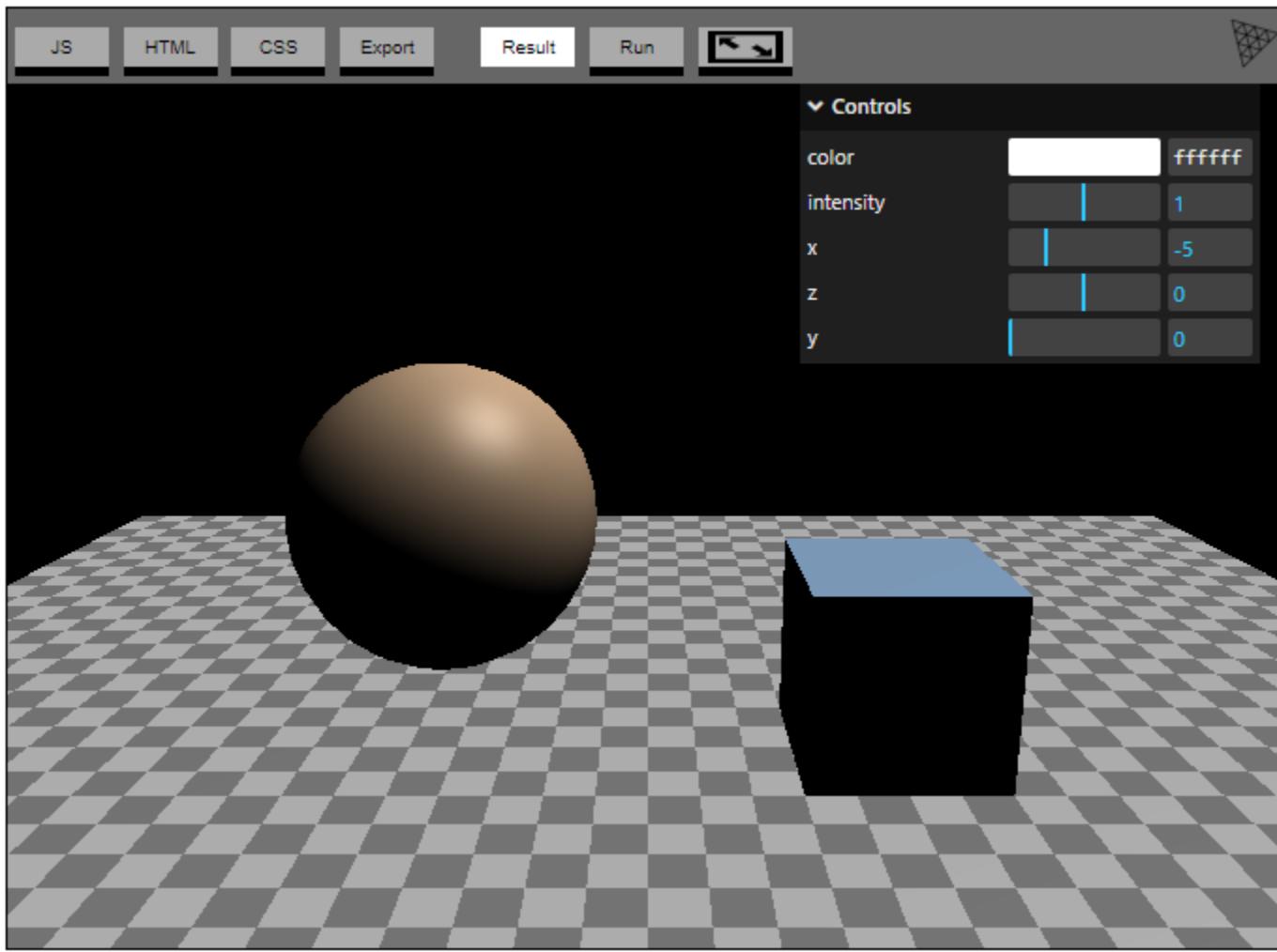
en ▾

Materials

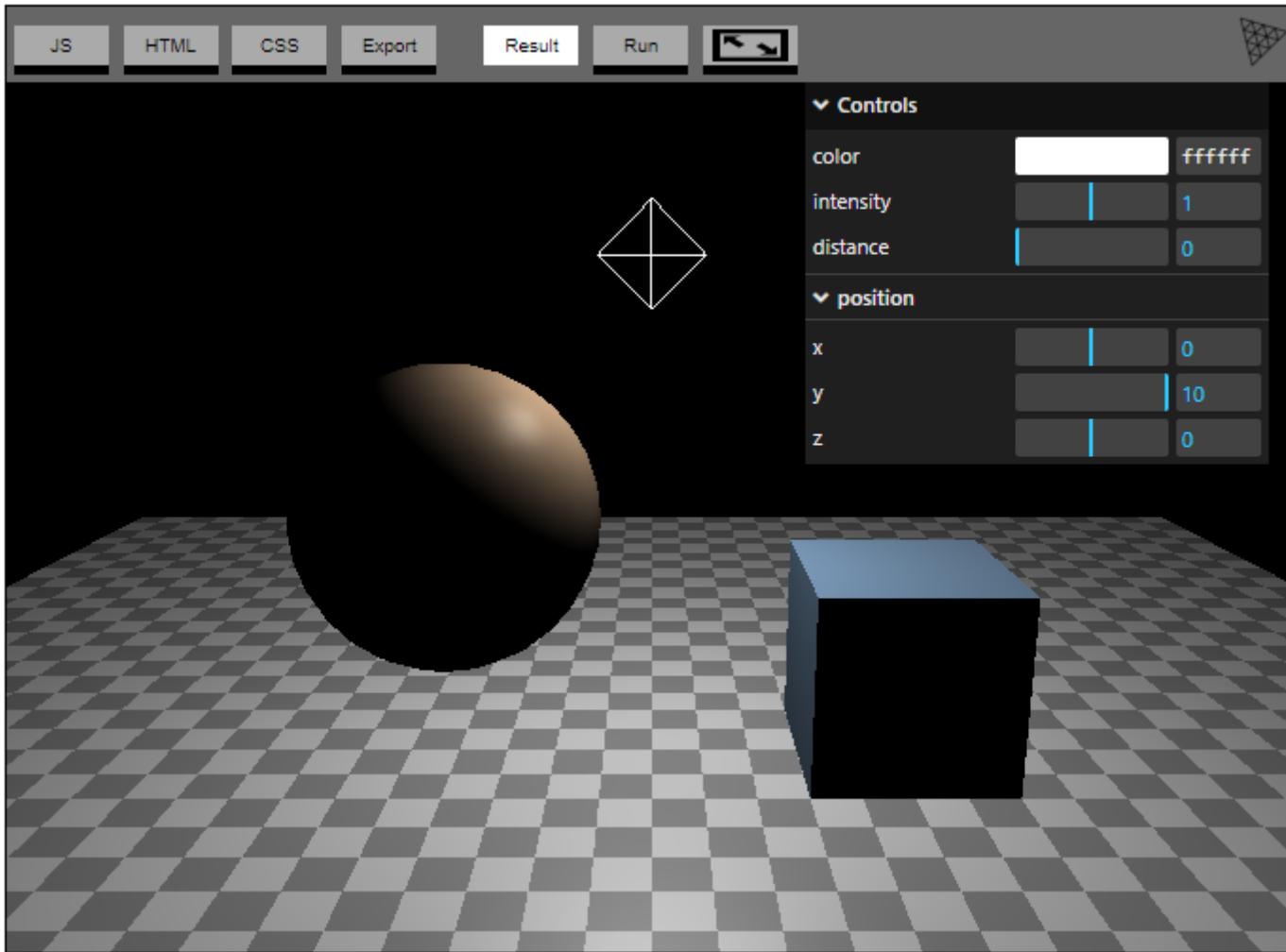
- [LineBasicMaterial](#)
- [LineDashedMaterial](#)
- [Material](#)
- [MeshBasicMaterial](#)
- [MeshDepthMaterial](#)
- [MeshDistanceMaterial](#)
- [MeshLambertMaterial](#)
- [MeshMatcapMaterial](#)
- [MeshNormalMaterial](#)
- [MeshPhongMaterial](#)
- [MeshPhysicalMaterial](#)
- [MeshStandardMaterial](#)

LIGHT SOURCES & SHADOWS

Three.js – Directional light source



Three.js – Point light source

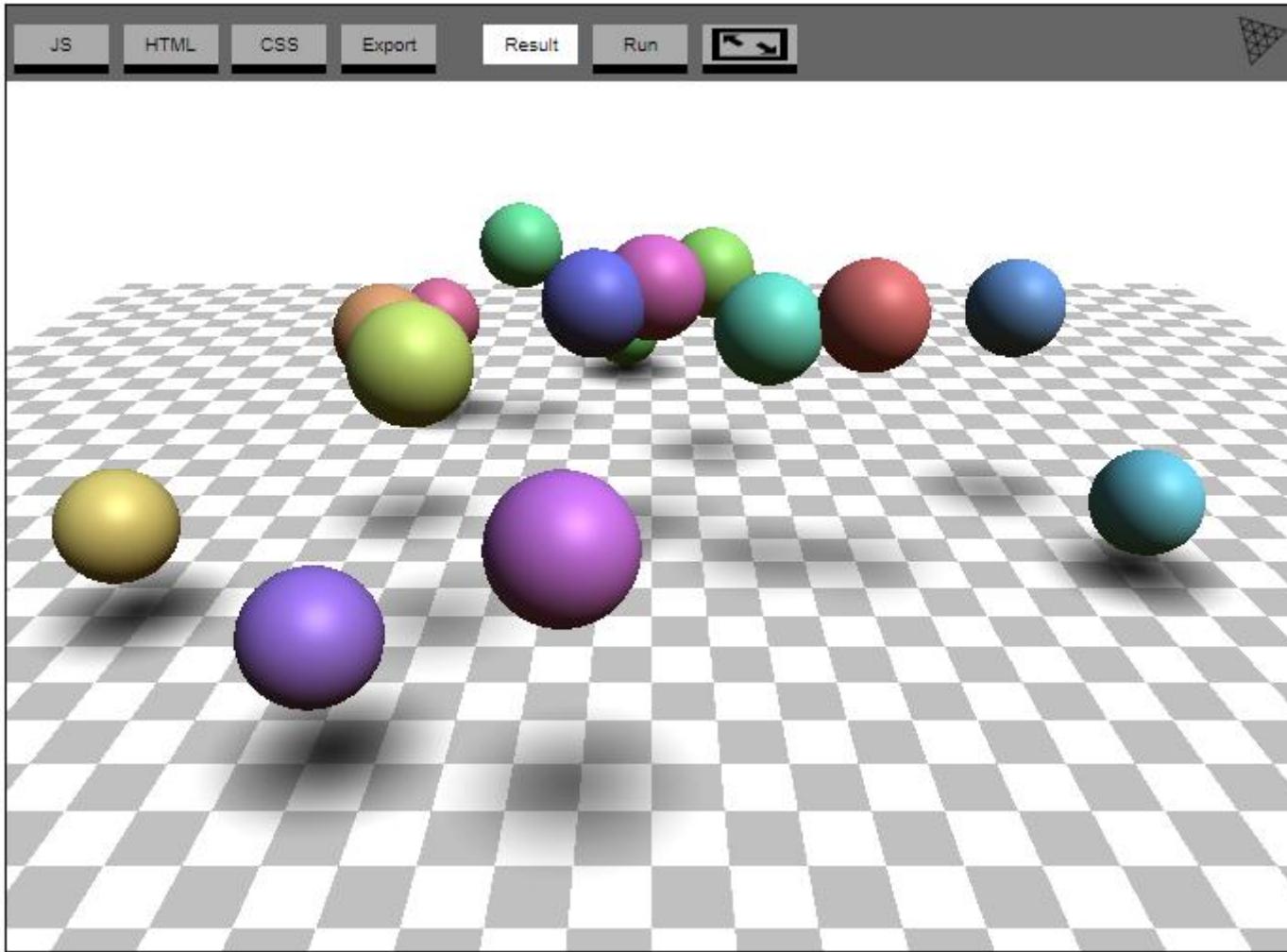


Three.js – Light sources

The screenshot shows a portion of the Three.js documentation website. At the top, there is a navigation bar with three items: "three.js" (highlighted in blue), "docs" (in grey), and "examples" (in grey). Below the navigation bar is a search bar with a magnifying glass icon and a language selector "en ▾". The main content area displays a list of light source classes under the heading "Lights". The listed classes are: AmbientLight, AmbientLightProbe, DirectionalLight, HemisphereLight, HemisphereLightProbe, Light, LightProbe, PointLight, RectAreaLight, and SpotLight.

- Lights**
- AmbientLight
- AmbientLightProbe
- DirectionalLight
- HemisphereLight
- HemisphereLightProbe
- Light
- LightProbe
- PointLight
- RectAreaLight
- SpotLight

Three.js - Shadows



Acknowledgments

- These slides are based on the first chapter of J. Dirksen's book: Learning Three.js
- And on the Three.js manual
- Thanks!

2D and 3D Transformations

Joaquim Madeira

March 2022

Topics

- Motivation
- 2D Transformations
- 3D Transformations
- Transformations in Three.js
- The Scene Graph

MOTIVATION

CG Main Tasks

■ Modeling

- Construct individual **models** / objects
- Assemble them into a 2D or 3D **scene**

■ Animation

- Static vs. dynamic scenes
- Movement and / or deformation

■ Rendering

- Generate final images
- Where is the viewer / camera ?
- How is he / she looking at the scene?

Modeling vs Rendering

■ Modeling

- Create models
- Apply materials to models
- Place models around scene
- Place lights in the scene
- Place the camera

[YouTube Demo](#)

■ Rendering

- Take picture with the camera

[van Dam]

Three.js – A scene with a cube

```
// The SCENE  
  
scene = new THREE.Scene();  
  
// The MODEL --- A cube is added to the scene  
  
var geometry = new THREE.BoxBufferGeometry( 200, 200, 200 );  
  
var material = new THREE.MeshBasicMaterial( { color: 0x00ff00 } );  
  
model = new THREE.Mesh( geometry, material );  
  
scene.add( model );
```



Three.js – The camera

```
// The CAMERA  
  
// --- Where the viewer is and how he is looking at the scene  
camera = new THREE.PerspectiveCamera( 70,  
                                      window.innerWidth / window.innerHeight, 1, 1000 );  
  
camera.position.z = 400;
```



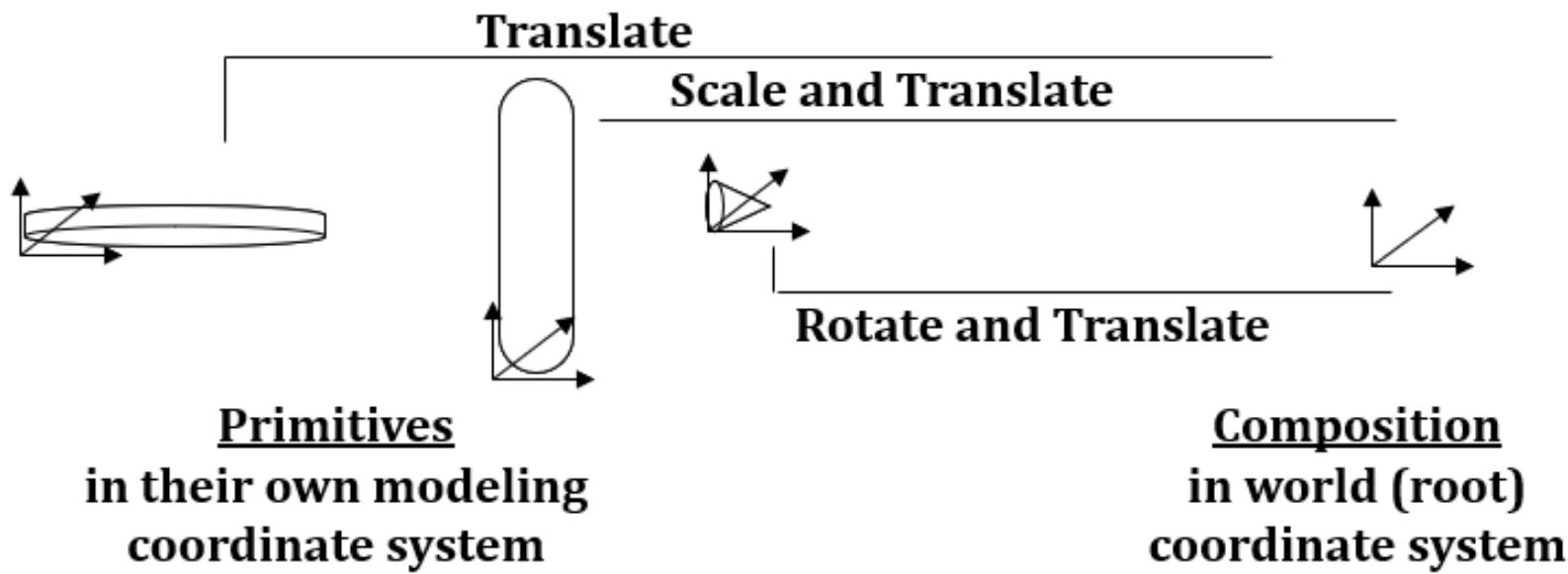
Three.js – Animation

```
function animate() {  
    requestAnimationFrame( animate );  
  
    model.rotation.x += 0.005;  
  
    model.rotation.y += 0.01;  
  
    renderer.render( scene, camera );  
}
```



Composition of a geometric model

- Assemble primitives to create final object
 - Apply affine transformations



[van Dam]

2D and 3D Transformations

- Position, rotate and scale objects on the 2D plane or in 3D space
- Basic transformations
 - Translation
 - Rotation
 - Scaling
- Matricial representation
 - Homogeneous coordinates !!
 - Concatenation = Matrix products
- Complex transformations ?
 - Decompose into a sequence of basic transformations

2D TRANSFORMATIONS

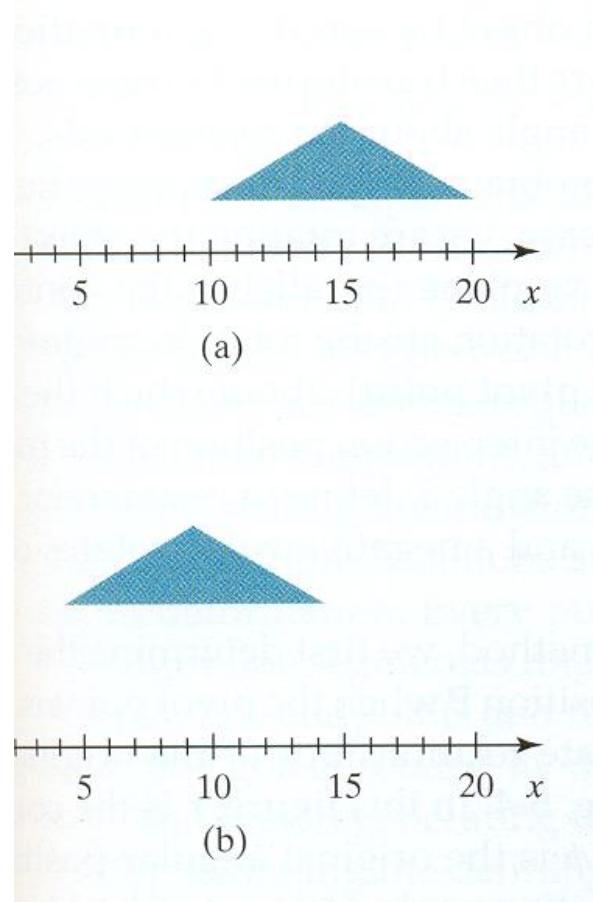
2D Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

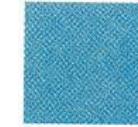
↑ ↓

$$\mathbf{P}' = \mathbf{T}(t_x, t_y) \cdot \mathbf{P}$$

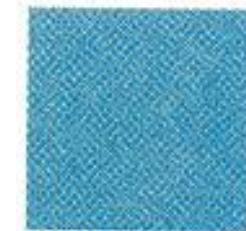
↑



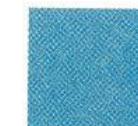
2D Scaling



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$



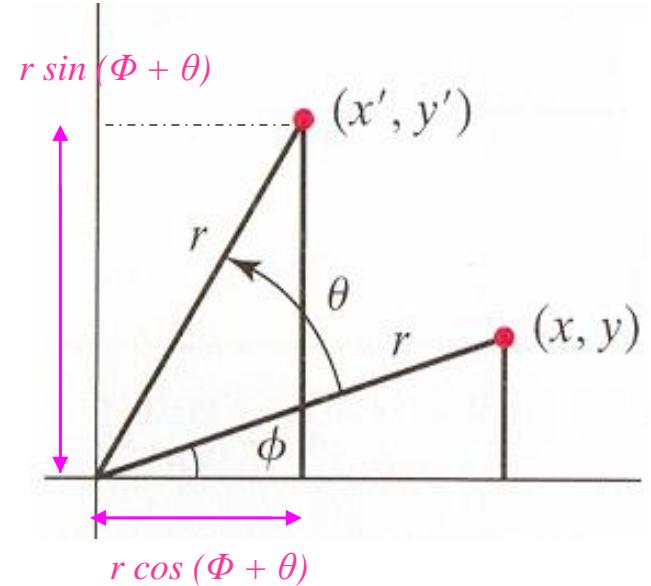
$$\mathbf{P}' = \mathbf{S}(s_x, s_y) \cdot \mathbf{P}$$



2D Rotation

$$x' = r \cos(\phi + \theta) = r \cos \phi \cos \theta - r \sin \phi \sin \theta$$

$$y' = r \sin(\phi + \theta) = r \cos \phi \sin \theta + r \sin \phi \cos \theta$$



$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

$$\mathbf{P}' = \mathbf{R}(\theta) \cdot \mathbf{P}$$

Concatenation

$$\begin{aligned}\mathbf{P}' &= \mathbf{T}(t_{2x}, t_{2y}) \cdot \{\mathbf{T}(t_{1x}, t_{1y}) \cdot \mathbf{P}\} \\ &= \{\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y})\} \cdot \mathbf{P}\end{aligned}$$

$$\begin{bmatrix} 1 & 0 & t_{2x} \\ 0 & 1 & t_{2y} \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & t_{1x} \\ 0 & 1 & t_{1y} \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & t_{1x} + t_{2x} \\ 0 & 1 & t_{1y} + t_{2y} \\ 0 & 0 & 1 \end{bmatrix}$$

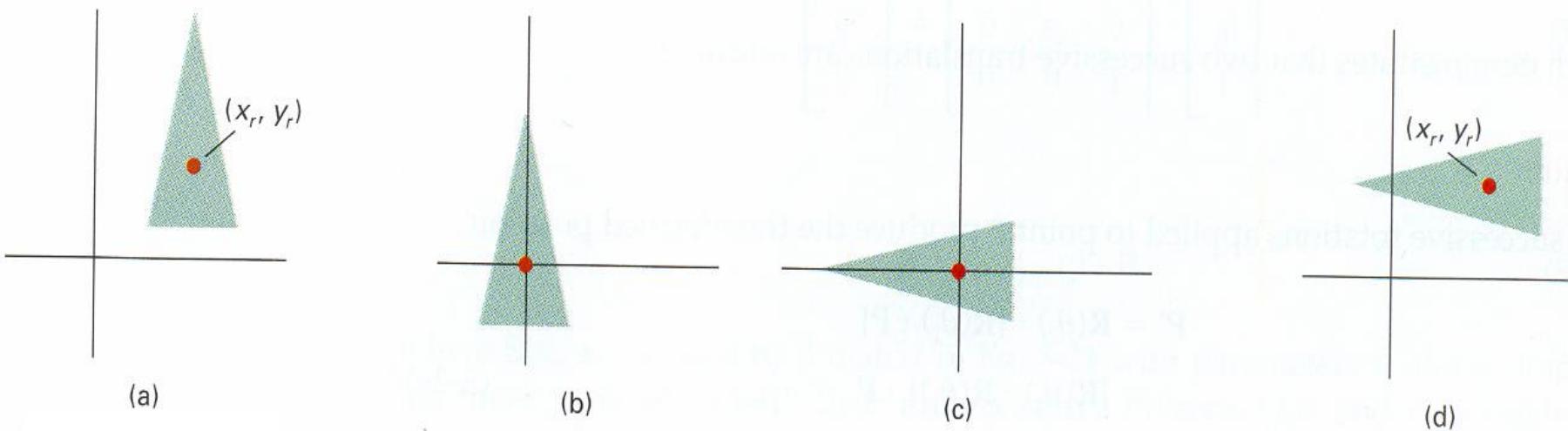
$$\mathbf{T}(t_{2x}, t_{2y}) \cdot \mathbf{T}(t_{1x}, t_{1y}) = \mathbf{T}(t_{1x} + t_{2x}, t_{1y} + t_{2y})$$

Concatenation

$$\begin{bmatrix} s_{2x} & 0 & 0 \\ 0 & s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} s_{1x} & 0 & 0 \\ 0 & s_{1y} & 0 \\ 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} s_{1x} \cdot s_{2x} & 0 & 0 \\ 0 & s_{1y} \cdot s_{2y} & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

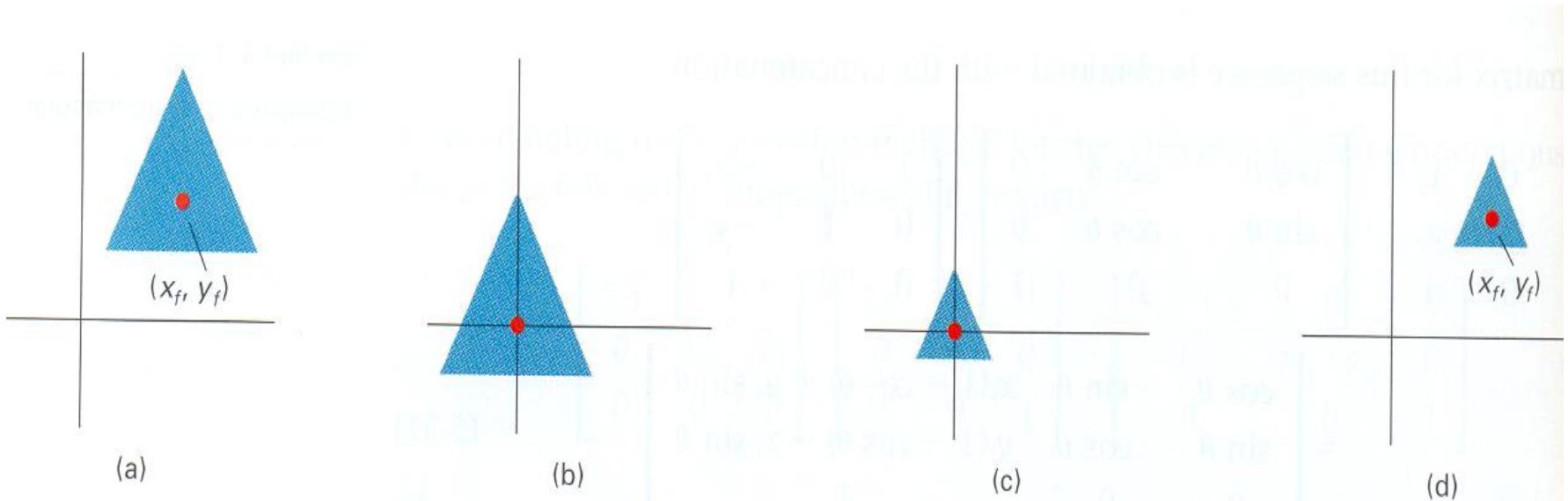
$$\mathbf{S}(s_{2x}, s_{2y}) \cdot \mathbf{S}(s_{1x}, s_{1y}) = \mathbf{S}(s_{1x} \cdot s_{2x}, s_{1y} \cdot s_{2y})$$

Arbitrary Rotation



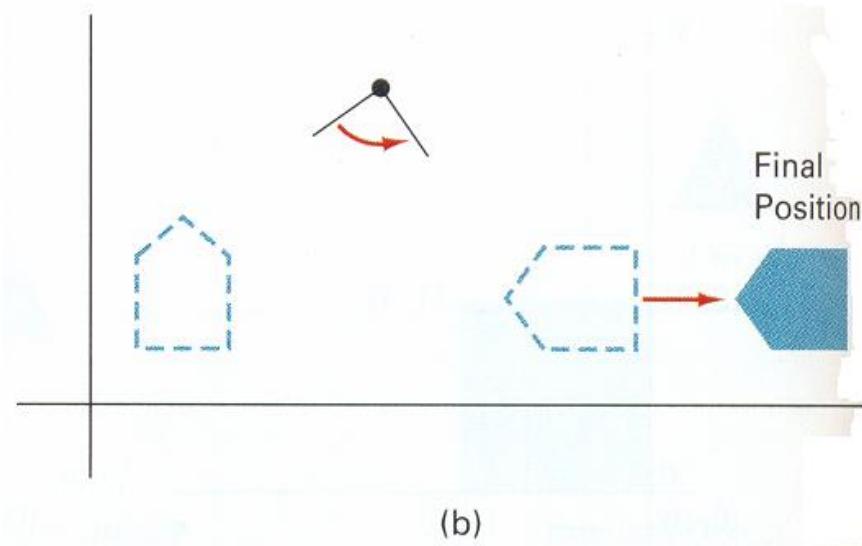
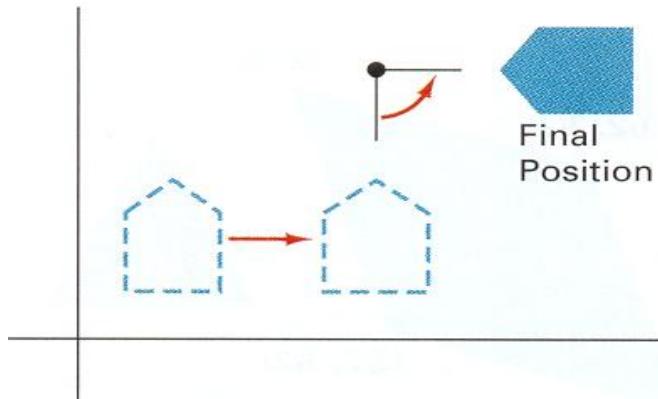
Translation + Rotation + Inverse Translation

Arbitrary Scaling



Translation + Scaling + Inverse Translation

Order is important !



3D TRANSFORMATIONS

3D Transformations

■ Translation

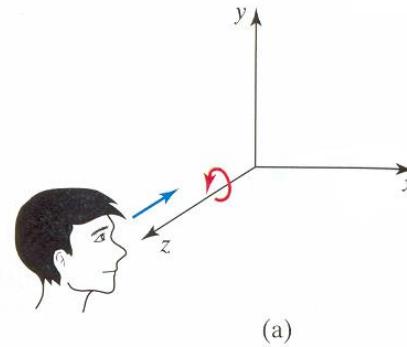
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & t_x \\ 0 & 1 & 0 & t_y \\ 0 & 0 & 1 & t_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

■ Scaling

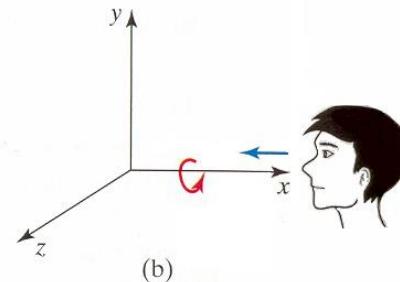
$$\mathbf{S} = \begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

3D Rotation

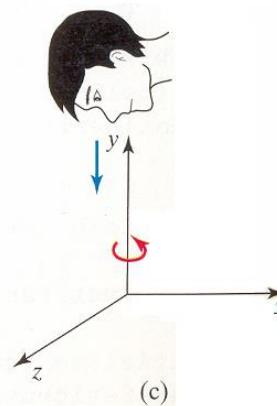
- Rotation around each one of the coordinate axis
- Positive rotations are CCW !!



(a)

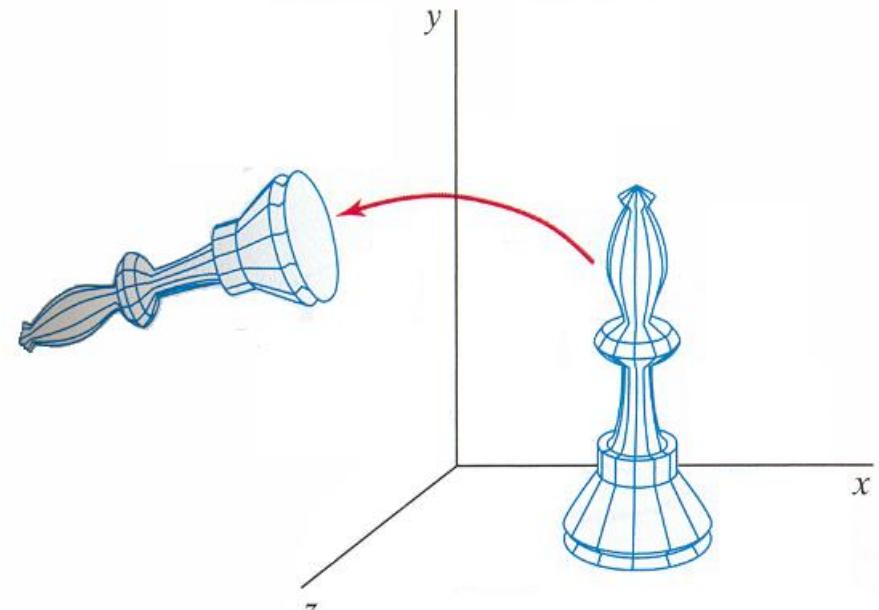


(b)



(c)

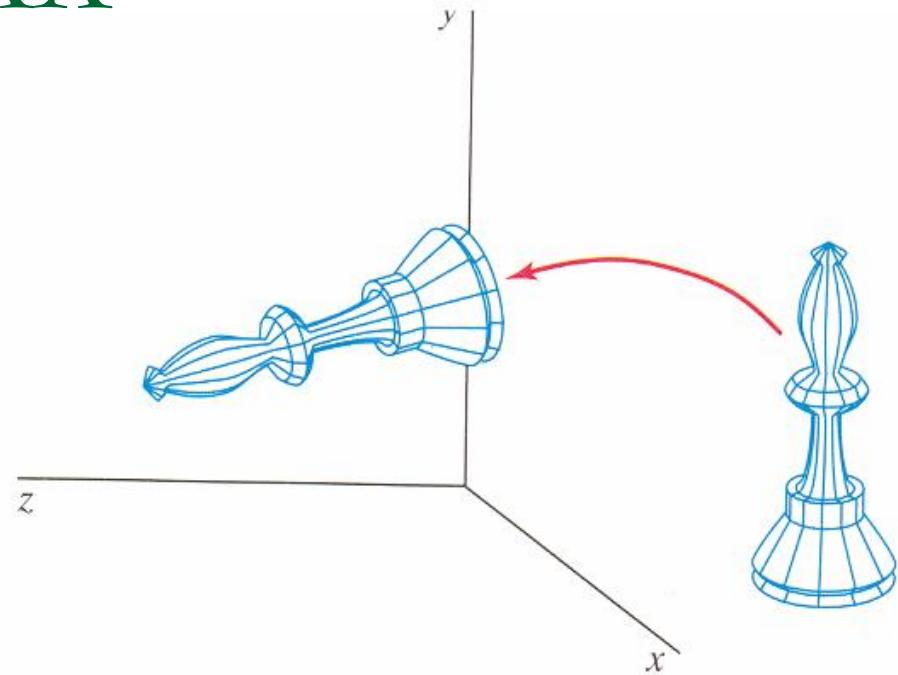
Rotation around ZZ'



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

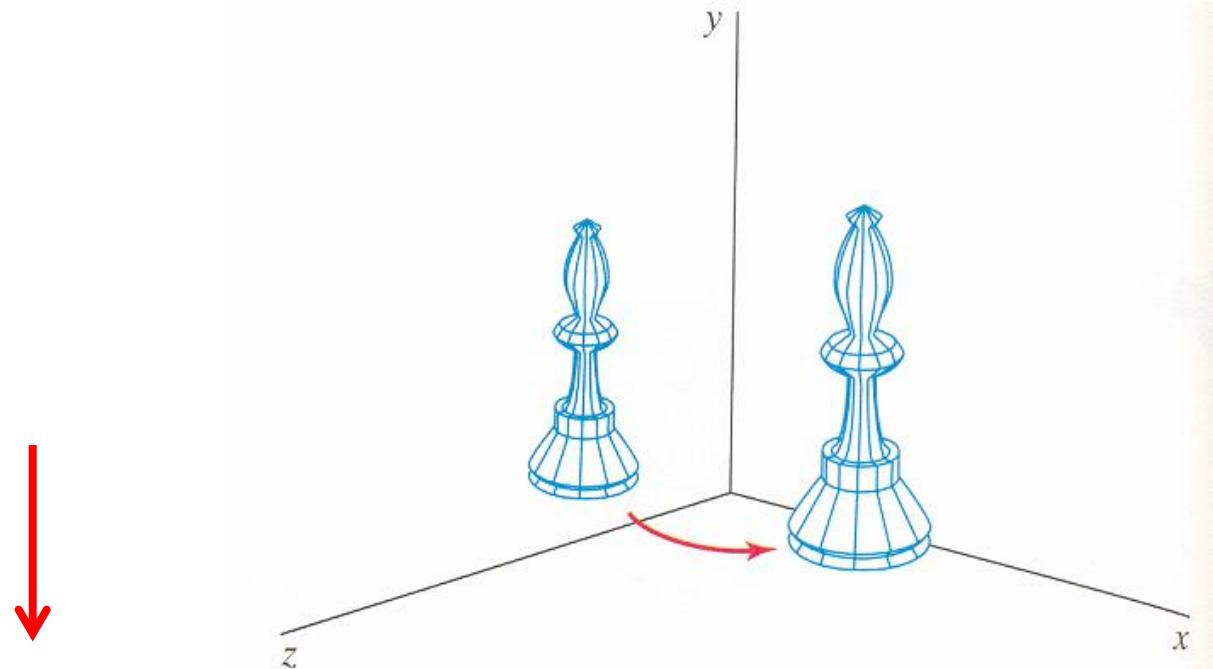


Rotation around XX'



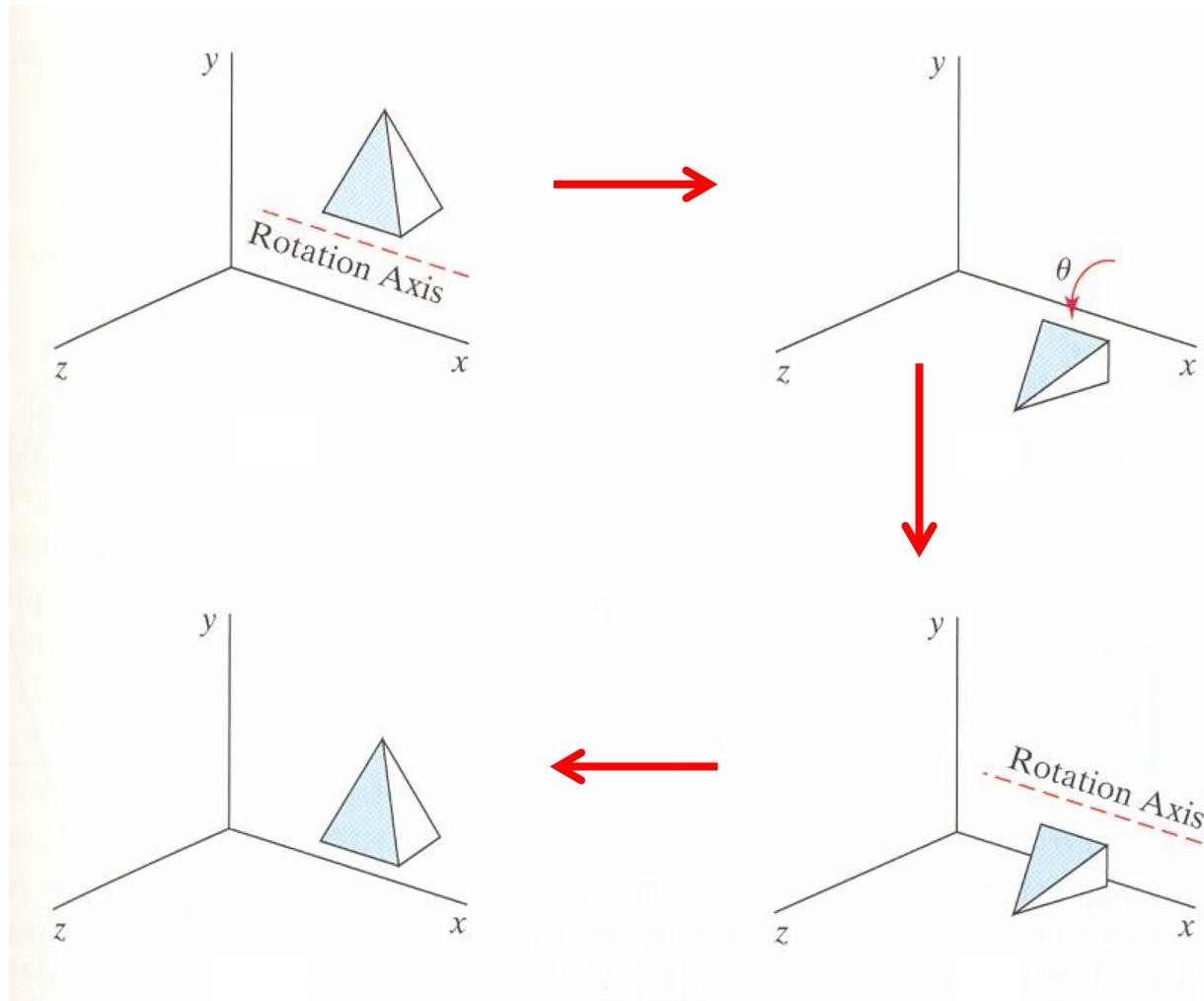
$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Rotation around YY'



$$\begin{bmatrix} x' \\ y' \\ z' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix}$$

Example – Decomposition



Task – Application Problems

- Solve the first application problems !!
- See if you have any questions / difficulties

Possible References

- 2D and 3D transformations are presented in any Computer Graphics book
- E. Angel and D. Shreiner. Interactive Computer Graphics, 7th Ed., Addison-Wesley, 2015
- J M Pereira, et al. Introdução à Computação Gráfica. FCA, 2018

TRANSFORMATIONS IN THREE.JS

Three.js – position

- Position of a model/object **relative to the position of its parent** in the scene graph
 - Default: (0,0,0)
- The parent is usually a **THREE.Scene** object or a **THREE.Object3D** object

```
cube.position.x=10;  
cube.position.y=3;  
cube.position.z=1;  
  
cube.position.set(10,3,1);  
  
cube.position=new THREE.Vector3(10,3,1)
```

Three.js – scale

- Scale factors of a model/object relative to its XX, YY or ZZ axis
 - Default: (1,1,1)

Three.js – rotation

- Rotation angle(s) for a model/object around any one of its **XX**, **YY** or **ZZ** axis
 - In radians !!
- And the **order** in which the rotations are carried out
 - Optional argument !!
- Default: (0,0,0,'**XYZ**')

```
cube.rotation.x = 0.5*Math.PI;  
cube.rotation.set(0.5*Math.PI, 0, 0);  
cube.rotation = new THREE.Vector3(0.5*Math.PI, 0, 0);
```

Three.js – 4×4 Matrix Class

Matrix4

A class representing a 4×4 [matrix](#).

The most common use of a 4×4 matrix in 3D computer graphics is as a [Transformation Matrix](#). For an introduction to transformation matrices as used in WebGL, check out [this tutorial](#).

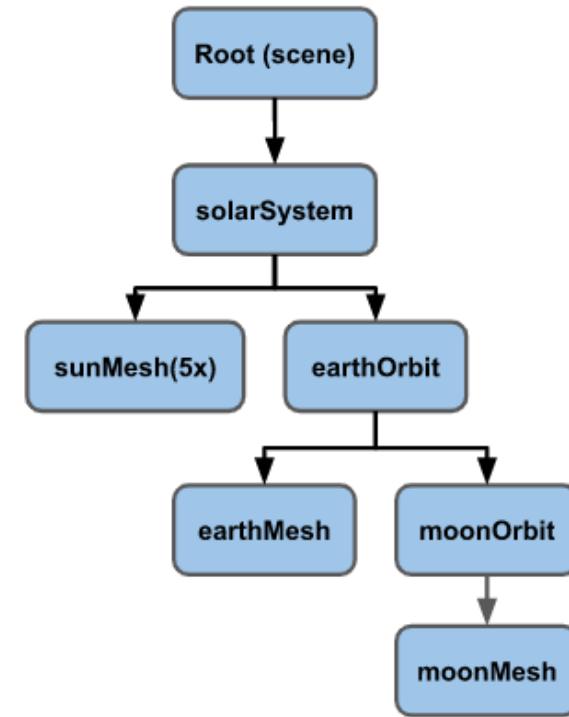
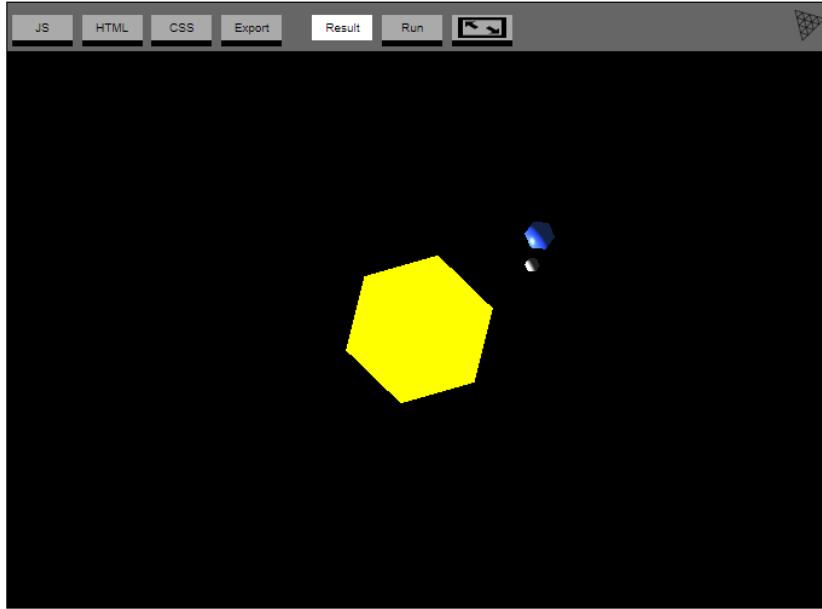
This allows a [Vector3](#) representing a point in 3D space to undergo transformations such as translation, rotation, shear, scale, reflection, orthogonal or perspective projection and so on, by being multiplied by the matrix. This is known as *applying* the matrix to the vector.

[<https://threejs.org/docs/#api/en/math/Matrix4>]

Local vs Global Transformations

- The **local transformation matrix** embodies the transformations defined on the **object own coordinate system**
 - `.matrix` attribute of a THREE.Object3D
- The **global transformation matrix** embodies **all transformations applied** to an object
 - `.matrixWorld` attribute of a THREE.Object3D
 - If the Object3D **has no parent**, it is identical to `.matrix`.

Local vs Global Transformations

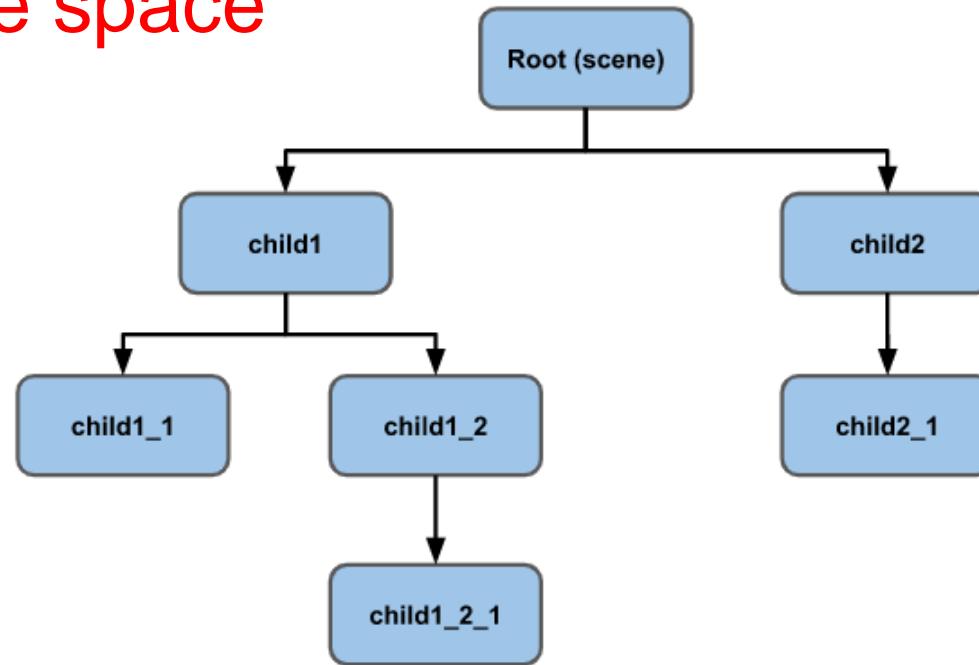


<https://threejs.org/manual/examples/scenegraph-sun-earth-moon.html>

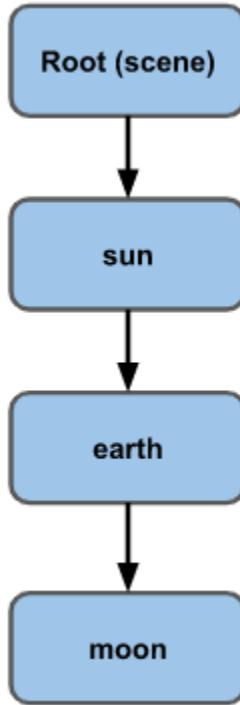
THE SCENE GRAPH

Scene Graph ?

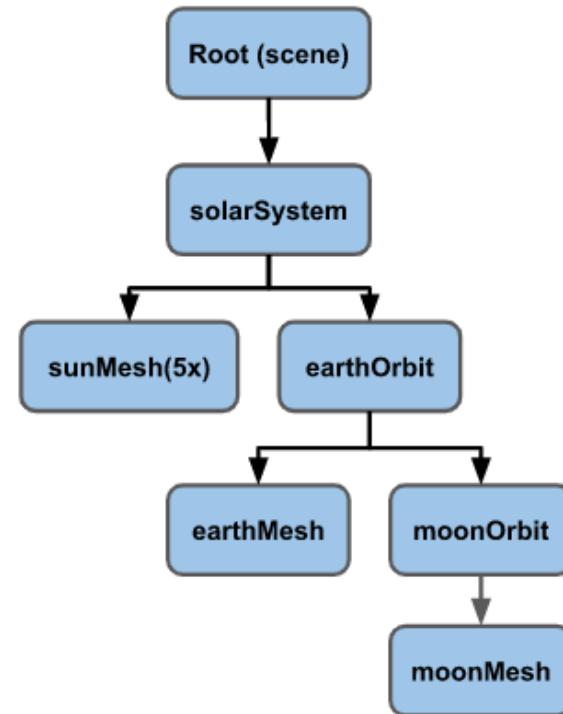
- A scene graph in a 3D engine is a **hierarchy of nodes**, where each node represents a **local coordinate space**



Simple solar system



?



The Sun

```
// an array of objects whose rotation to update
const objects = [];  
  
// use just one sphere for everything
const radius = 1;
const widthSegments = 6;
const heightSegments = 6;
const sphereGeometry = new THREE.SphereGeometry(
    radius, widthSegments, heightSegments);  
  
const sunMaterial = new THREE.MeshPhongMaterial({emissive: 0xFFFF00});
const sunMesh = new THREE.Mesh(sphereGeometry, sunMaterial);
sunMesh.scale.set(5, 5, 5); // make the sun large
scene.add(sunMesh);
objects.push(sunMesh);
```

The Earth

```
const earthMaterial = new THREE.MeshPhongMaterial({color: 0x2233FF, emissive: 0x112244});  
const earthMesh = new THREE.Mesh(sphereGeometry, earthMaterial);  
earthMesh.position.x = 10;    ←  
scene.add(earthMesh);        ←  
objects.push(earthMesh);
```

```
objects.forEach((obj) => {  
    obj.rotation.y = time;  
});
```

- Sun and Earth rotate around its own YY axis !!
 - How to improve ?

The Solar System – Objects ?

```
+ const solarSystem = new THREE.Object3D(); ←  
+ scene.add(solarSystem);  
+ objects.push(solarSystem);  
  
const sunMaterial = new THREE.MeshPhongMaterial({emissive: 0xFFFF00});  
const sunMesh = new THREE.Mesh(sphereGeometry, sunMaterial);  
sunMesh.scale.set(5, 5, 5);  
- scene.add(sunMesh);  
+ solarSystem.add(sunMesh); ←  
objects.push(sunMesh);  
  
const earthMaterial = new THREE.MeshPhongMaterial({color: 0x2233FF, emissive: 0x112244});  
const earthMesh = new THREE.Mesh(sphereGeometry, earthMaterial);  
earthMesh.position.x = 10;  
- sunMesh.add(earthMesh);  
+ solarSystem.add(earthMesh); ←  
objects.push(earthMesh);
```

The Earth Orbit

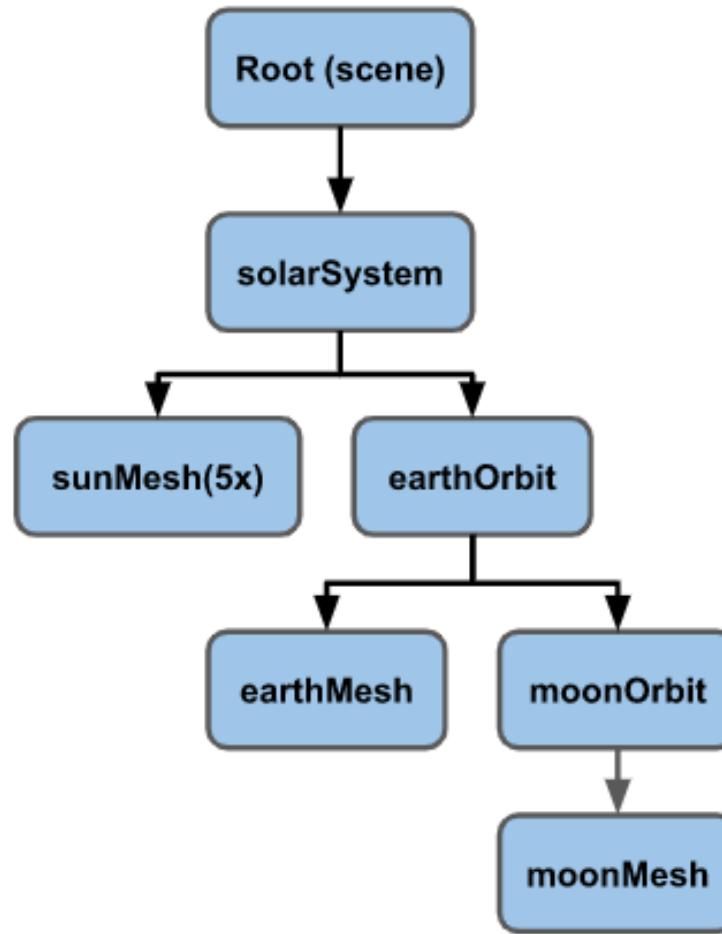
```
+ const earthOrbit = new THREE.Object3D(); ←
+ earthOrbit.position.x = 10; ←
+ solarSystem.add(earthOrbit);
+ objects.push(earthOrbit);

const earthMaterial = new THREE.MeshPhongMaterial({color: 0x2233FF, emissive: 0x112244});
const earthMesh = new THREE.Mesh(sphereGeometry, earthMaterial);
- earthMesh.position.x = 10; // note that this offset is already set in its parent's THREE
- solarSystem.add(earthMesh);
+ earthOrbit.add(earthMesh); ←
objects.push(earthMesh);
```

The Moon Orbit

```
+ const moonOrbit = new THREE.Object3D(); ←  
+ moonOrbit.position.x = 2; ←  
+ earthOrbit.add(moonOrbit);  
  
+ const moonMaterial = new THREE.MeshPhongMaterial({color: 0x888888, emissive: 0x222222});  
+ const moonMesh = new THREE.Mesh(sphereGeometry, moonMaterial);  
+ moonMesh.scale.set(.5, .5, .5); ←  
+ moonOrbit.add(moonMesh); ←  
+ objects.push(moonMesh);
```

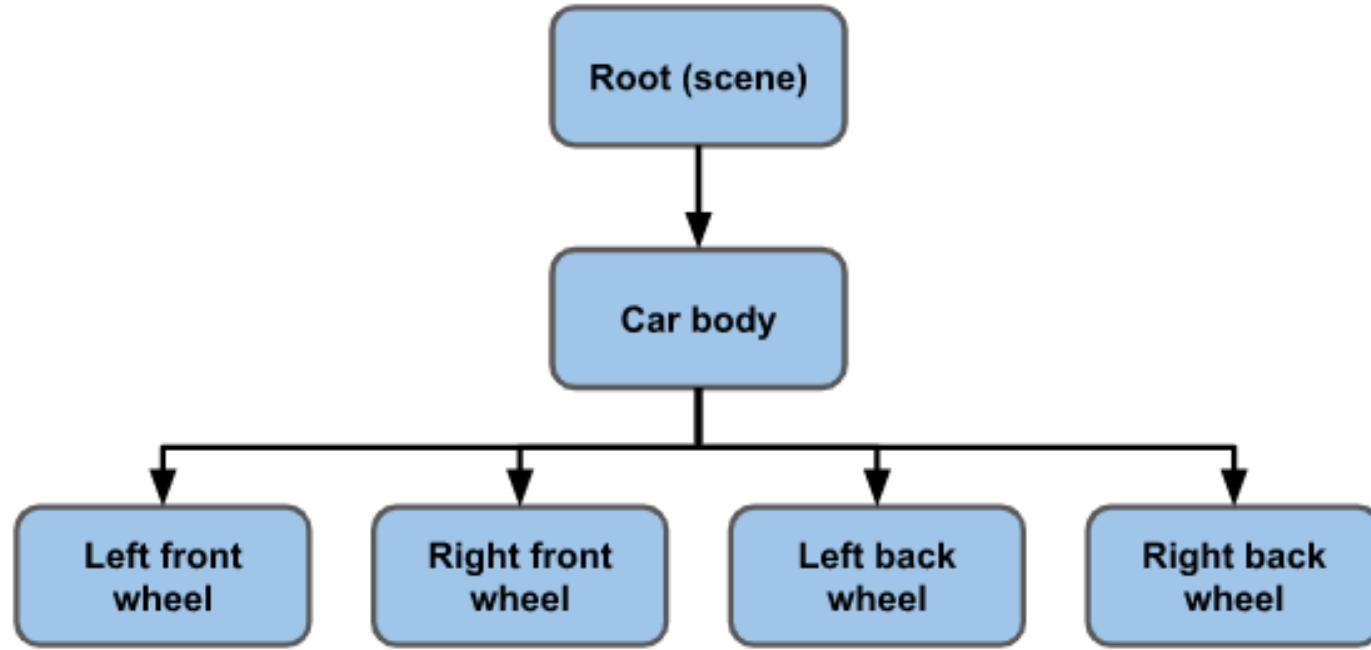
The Scene Graph



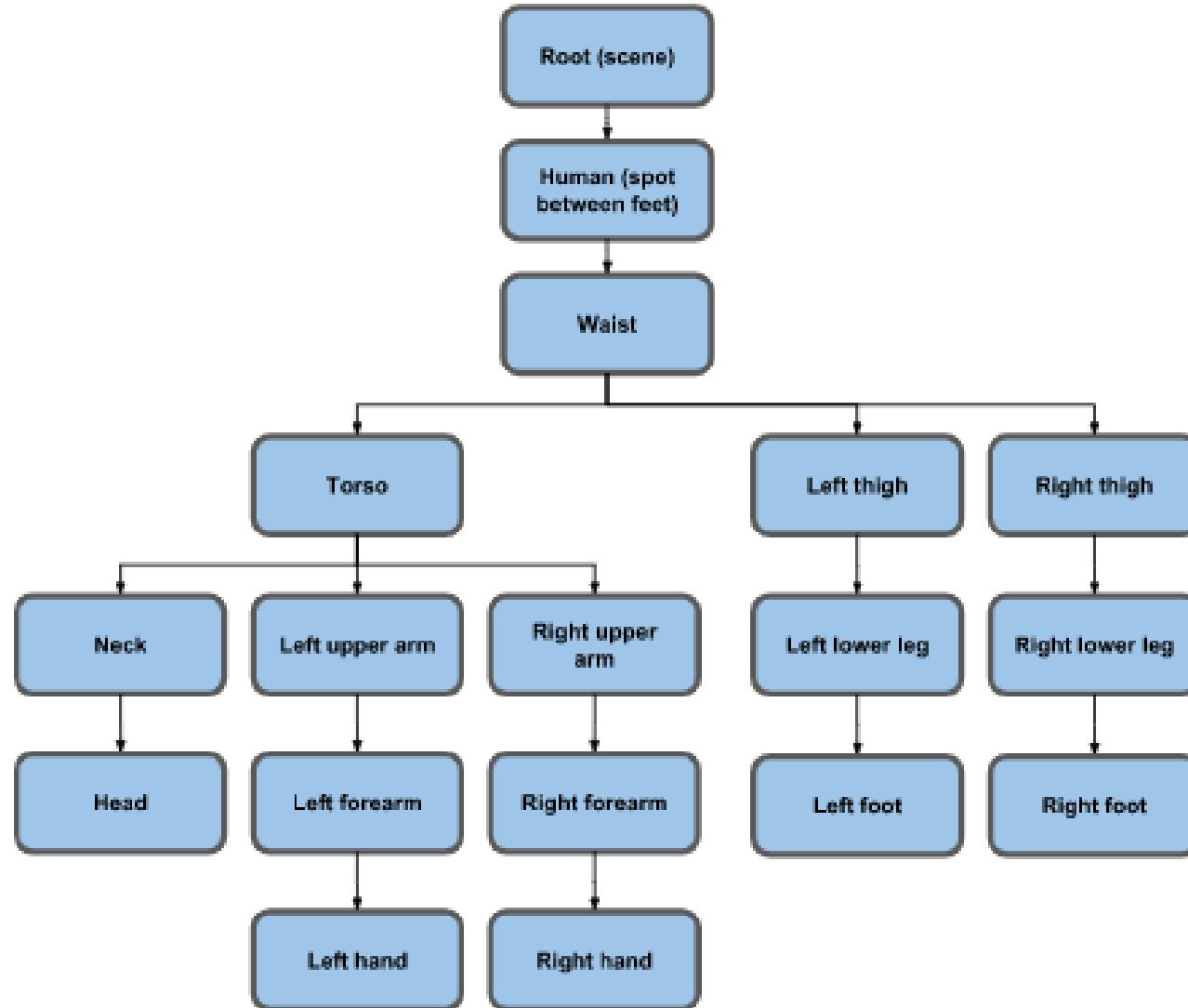
Task

- Revise the solar system exemple
<https://threejs.org/manual/#en/scenegraph>
- See if you have any questions / difficulties !!

Another exemple – A car



Example – A human in a game world



Acknowledgment

- Example code and figures taken from
<https://threejs.org/manual/#en/scenegraph>

3D Viewing

Joaquim Madeira

March 2022

Topics

- Recap
- Projections
- Matricial Representation
- View Volume & Clipping
- Visible-Surface Determination
- Three.js – Cameras

RECAP

CG Main Tasks

■ Modeling

- Construct individual **models** / objects
- Assemble them into a 2D or 3D **scene**

■ Animation

- Static vs. dynamic scenes
- Movement and / or deformation

■ Rendering

- Generate final images
- Where is the viewer / camera ?
- How is he / she looking at the scene?



Modeling vs Rendering

■ Modeling

- Create models
- Apply materials to models
- Place models around scene
- Place lights in the scene
- Place the camera

YouTube Demo



■ Rendering

- Take picture with the camera



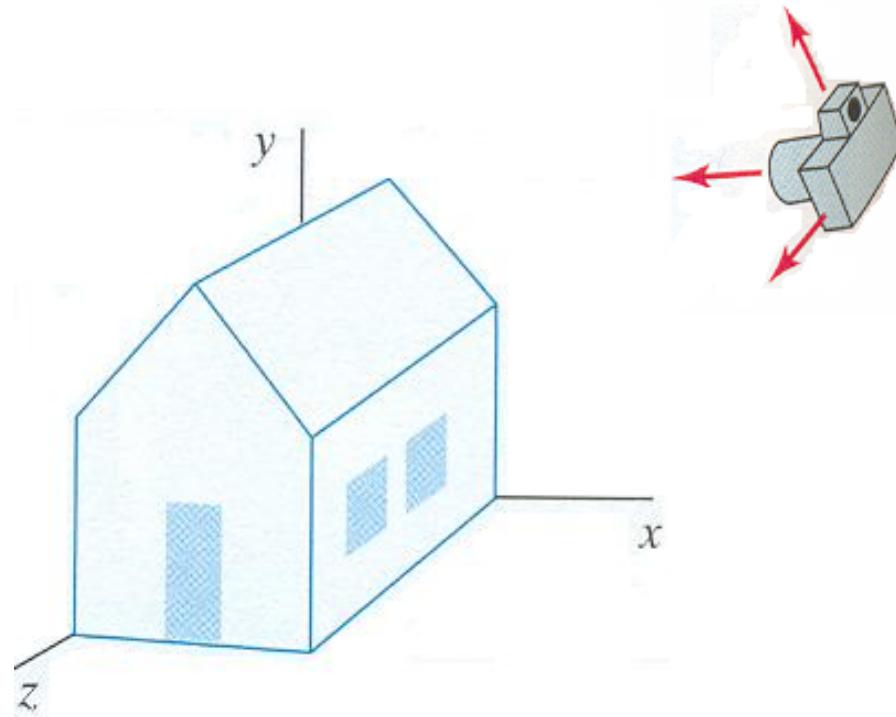
[van Dam]

Transformations

- Position, rotate and scale objects
- Basic transformations
 - Translation
 - Rotation
 - Scaling
- Matricial representation
 - Homogeneous coordinates !! ←
 - Concatenation = Matrix products ←
- Complex transformations ?
 - Decompose into a sequence of basic transformations

3D VIEWING

3D Viewing



3D Viewing

- Where is the observer / the camera ?
 - Position ?
 - Close to the 3D scene ?
 - Far away ?
- How is the observer looking at the scene ?
 - Orientation ?
- How to represent as a 2D image ?
 - Projection ?

Three.js – The camera

```
// The CAMERA  
  
// --- Where the viewer is and how he is looking at the scene  
  
camera = new THREE.PerspectiveCamera( 70,  
                                         window.innerWidth / window.innerHeight, 1, 1000 );  
  
camera.position.z = 400;
```

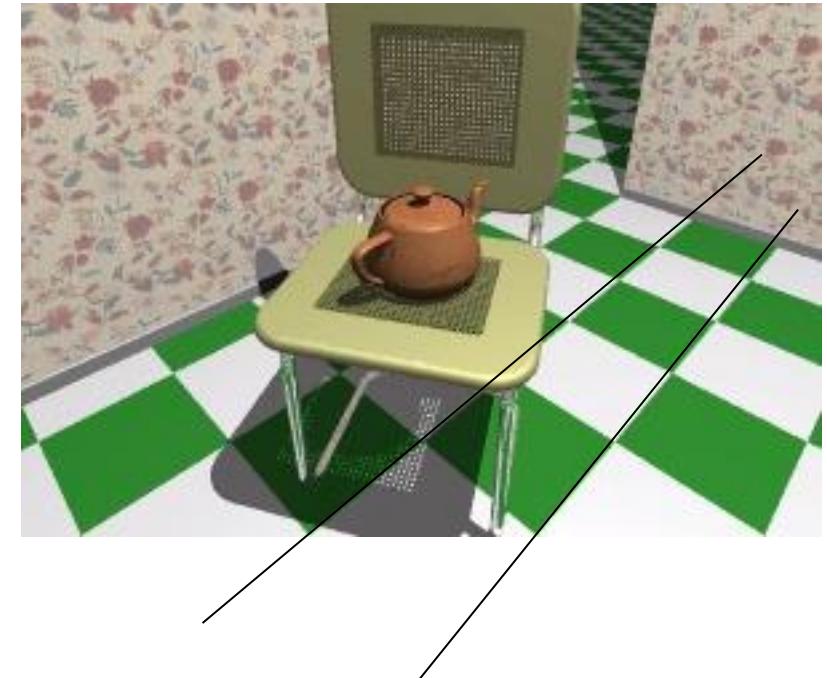


PROJECTIONS

Projections

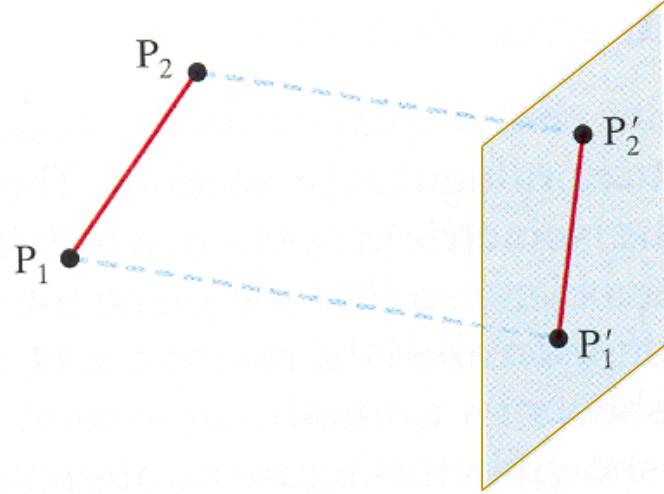


Parallel Projection

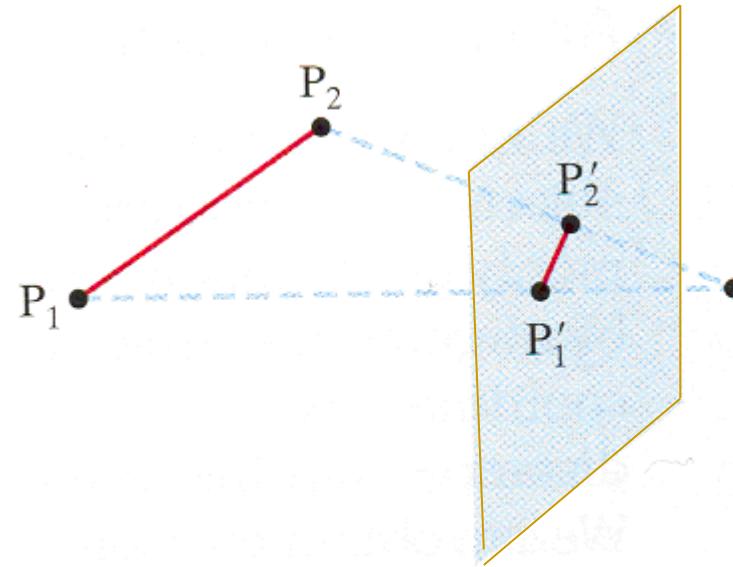


Perspective Projection

Projections

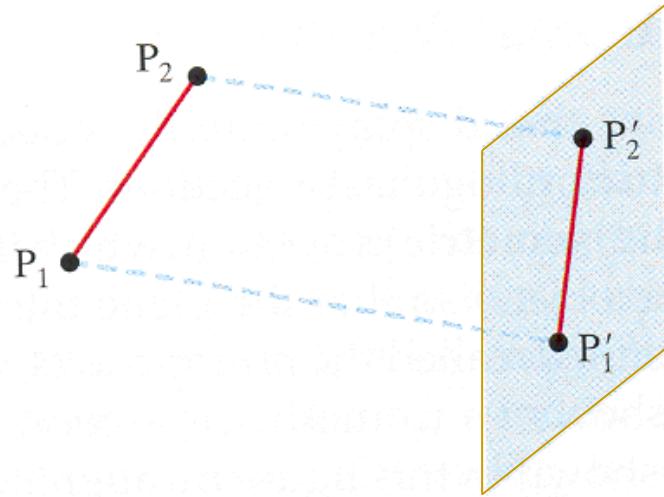


Parallel Projection

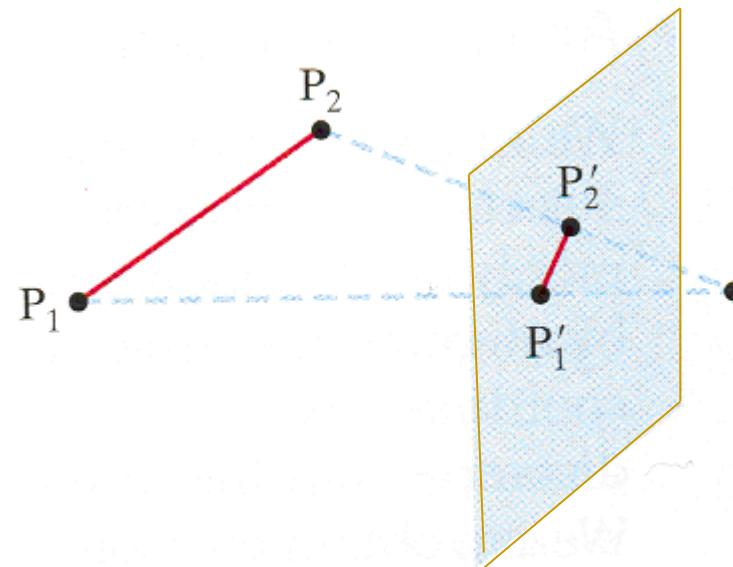


Perspective Projection

Projections

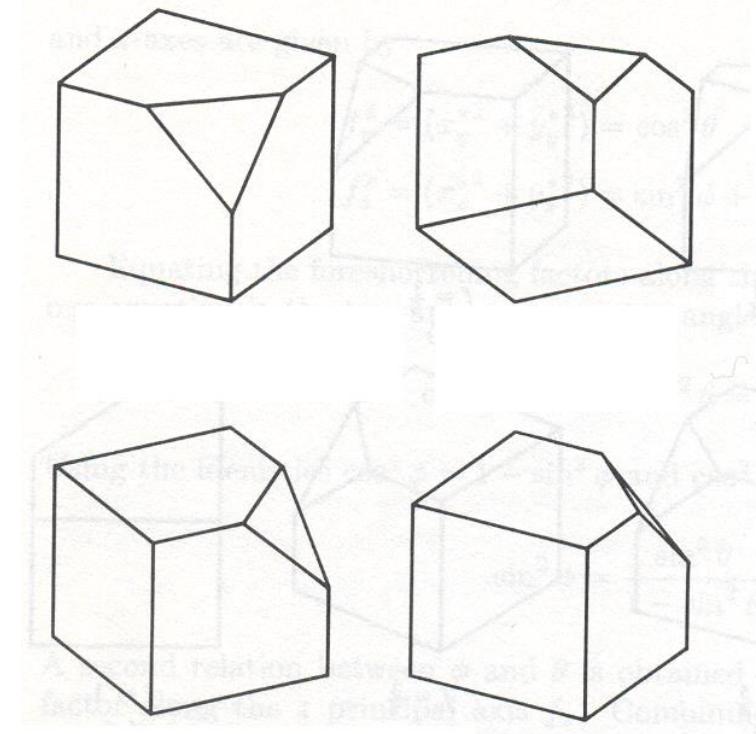
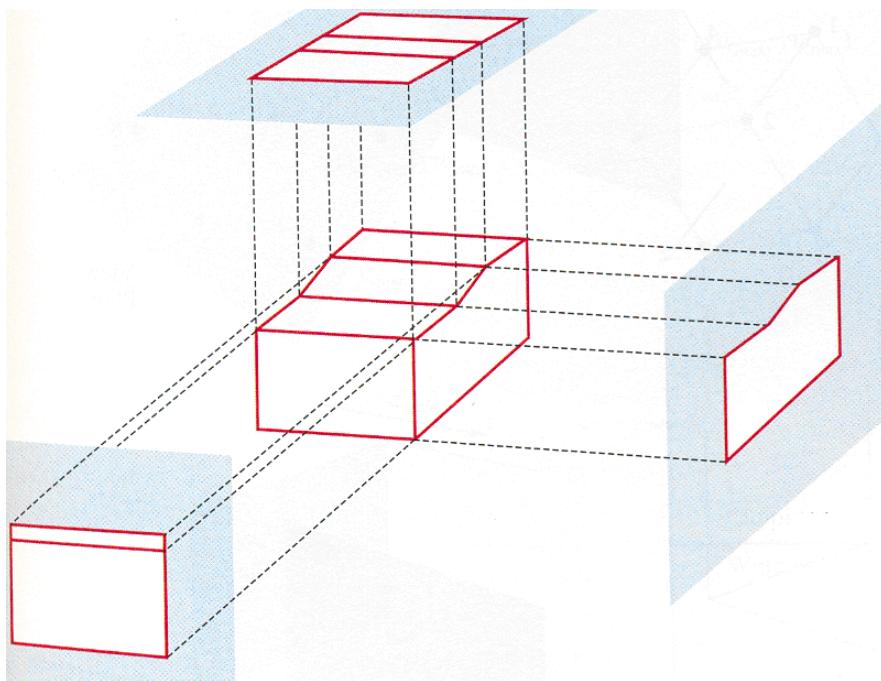


The projector straight-lines are **parallel**, i.e., converge at an **indefinite distance**

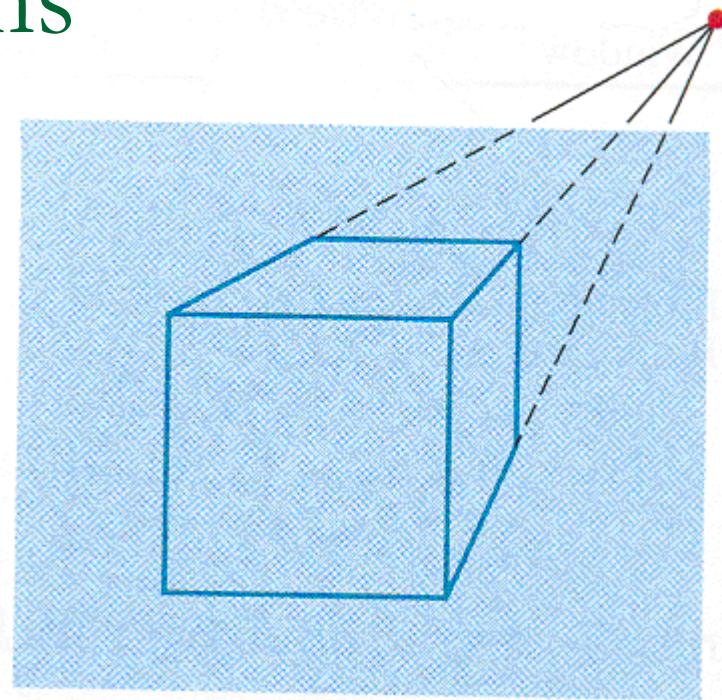
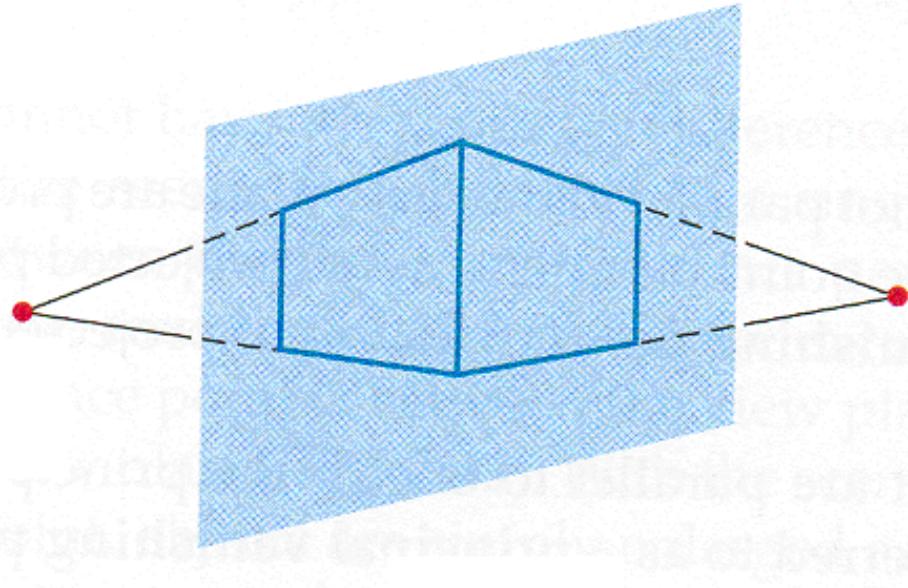


The projector straight-lines converge at the **projection center**

Parallel Projections



Perspective Projections



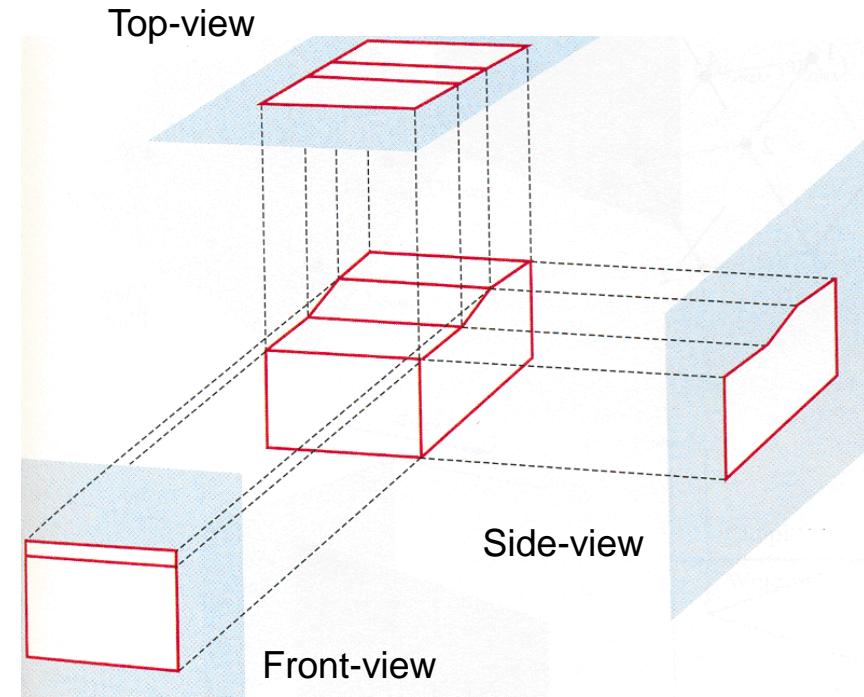
How to represent ?

- Projection **matrices**
- Homogeneous coordinates
- Concatenation through matrix multiplication
- Don't worry !
- Graphics APIs implement usual projections !

ORTHOGONAL PARALLEL PROJECTIONS

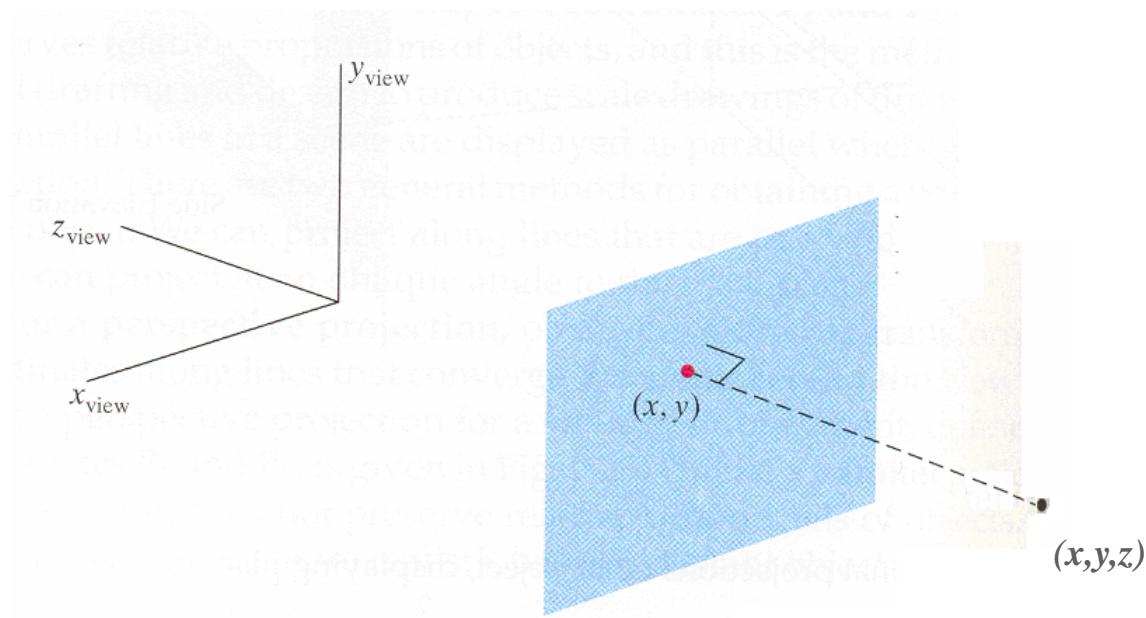
Orthogonal Parallel Projections

- The **projectors** are **perpendicular** to the projection plane
- The **projection plane** is **parallel** to a set of the object's faces
- Some **angles, lengths and areas** can be directly measured
- The views might not convey the 3D structure / shape of the objects
- Frequently used in Engineering and Architecture

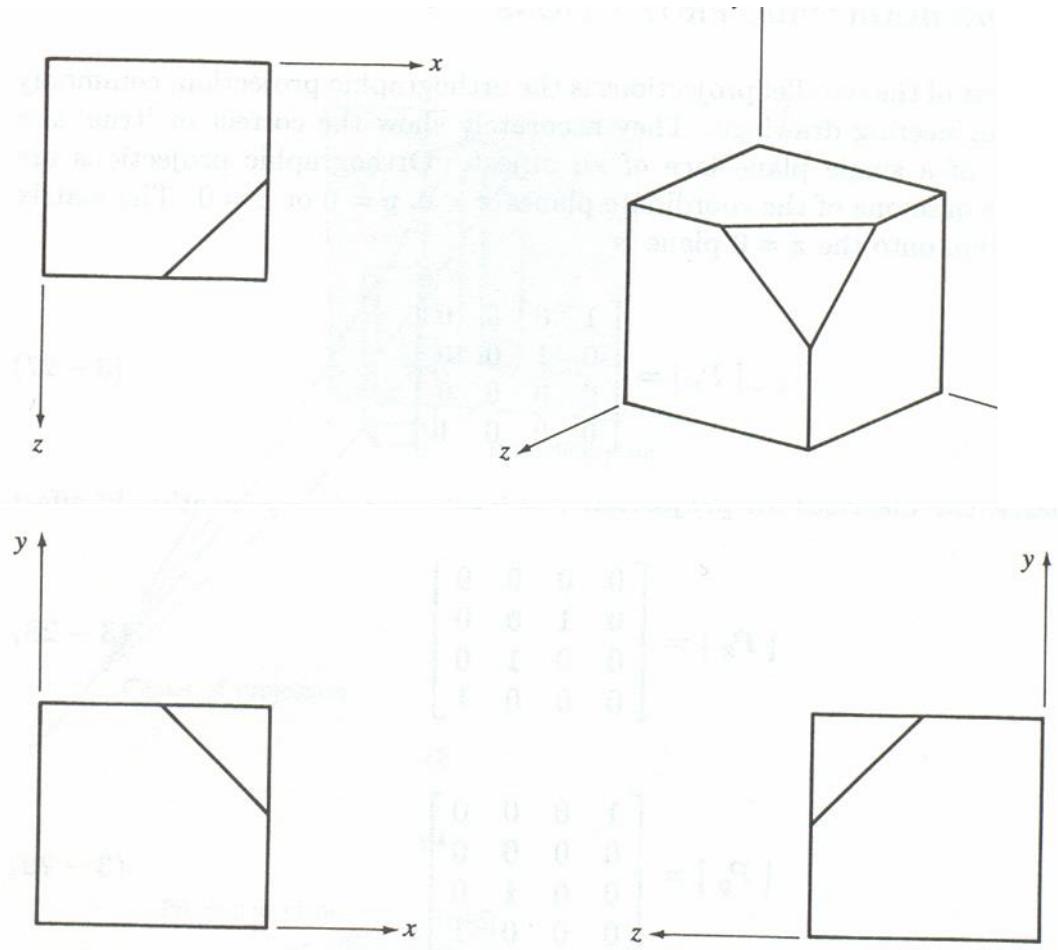


Parallel Projection Coordinates

- If the direction projection is parallel to the ZZ' axis, what are the **coordinates of the projected point** ?

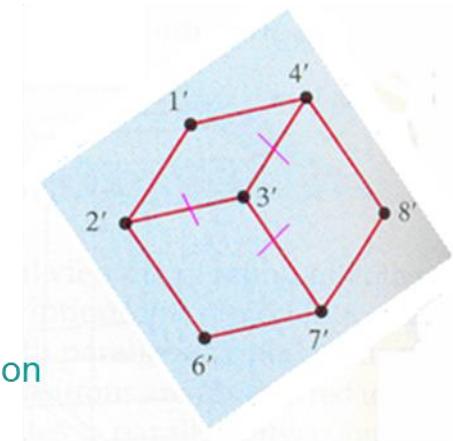
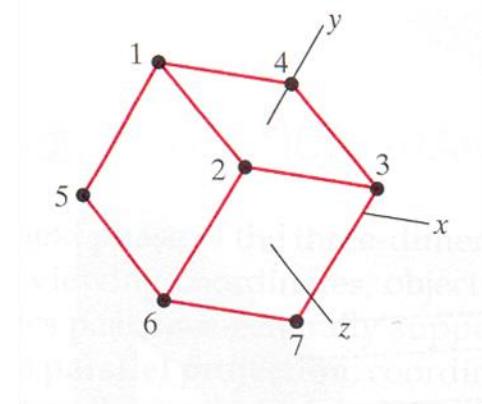


Orthogonal Parallel Projections



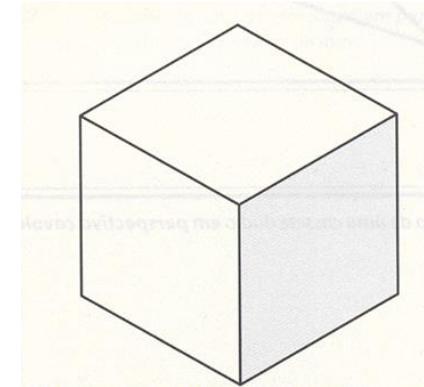
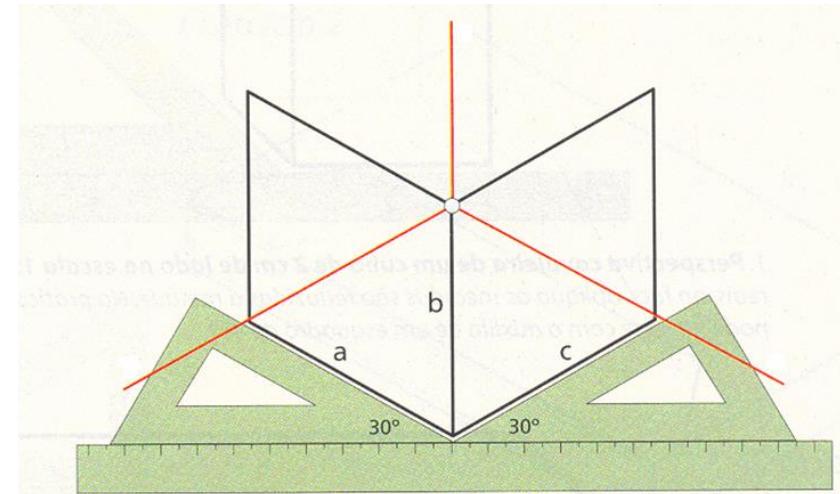
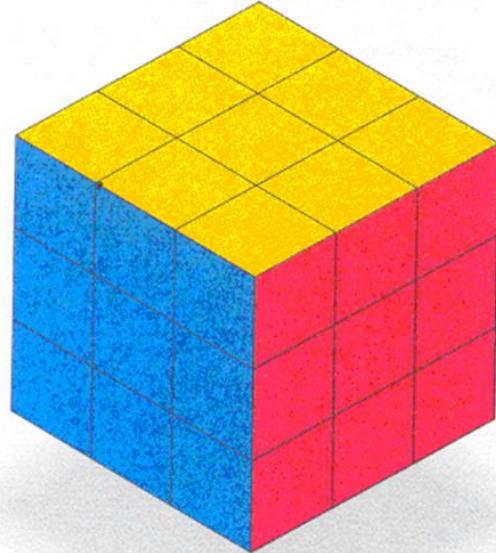
Axonometric Projections

- Orthogonal parallel projections, where the projection plane is **not parallel** to a set of the object's faces
- Give a better idea of the object's 3D structure / shape
- 3 classes
 - Isometric
 - Dimetric
 - Trimetric

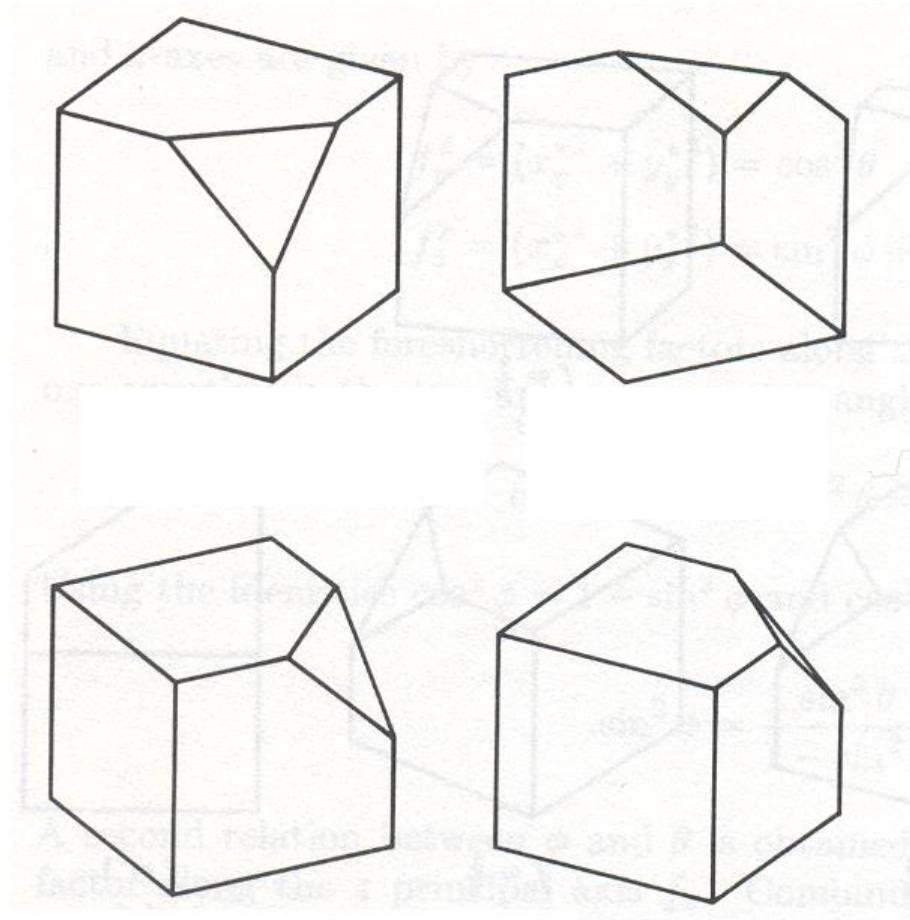


Isometric projection of a cube:
3 faces are shown and all edges
have the same length

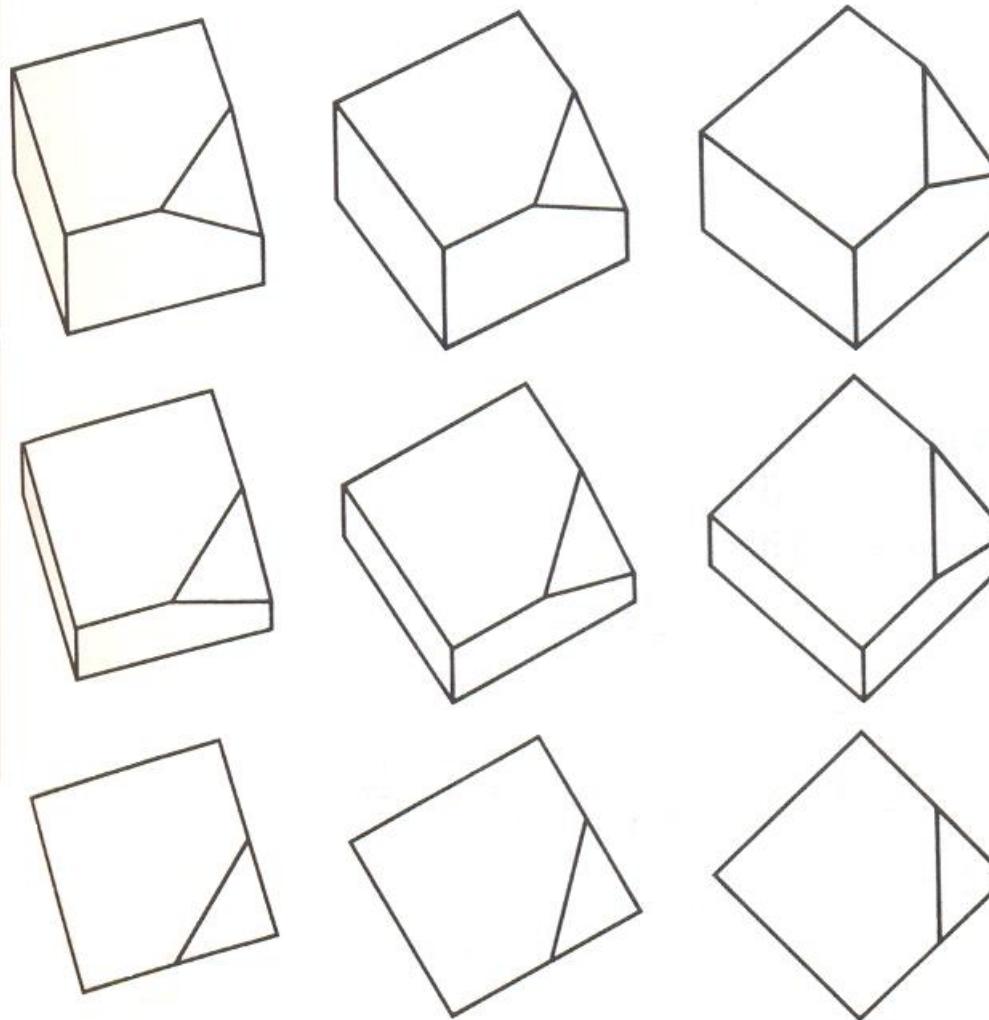
Drawing an isometric projection



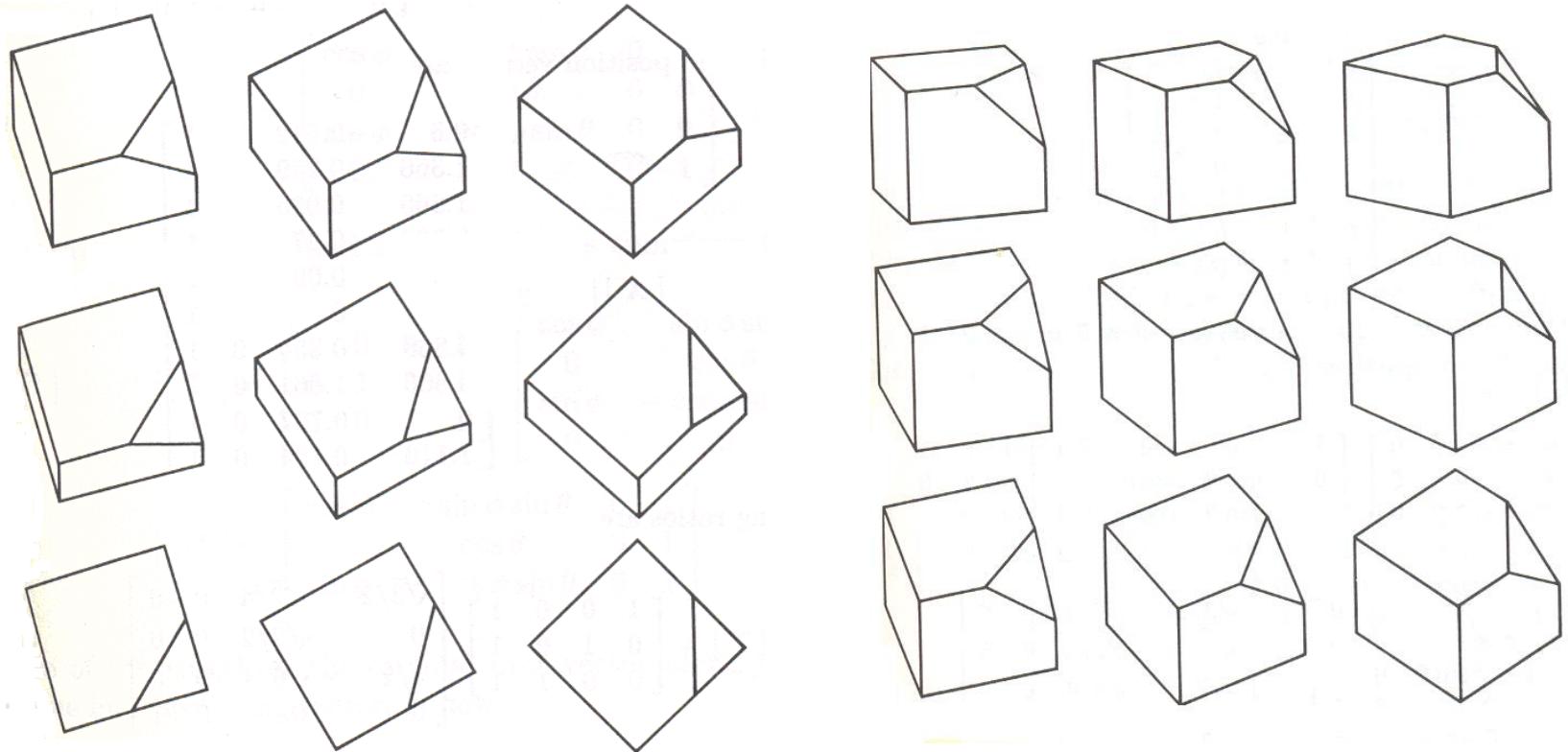
Isometric Projections



Dimetric Projections



Trimetric Projections



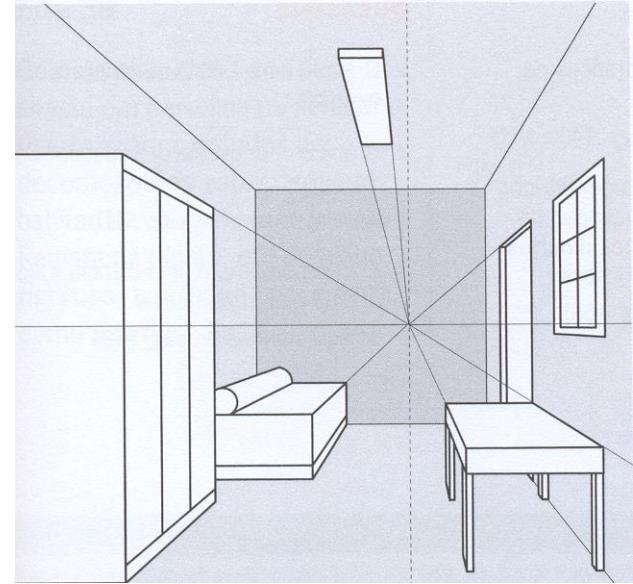
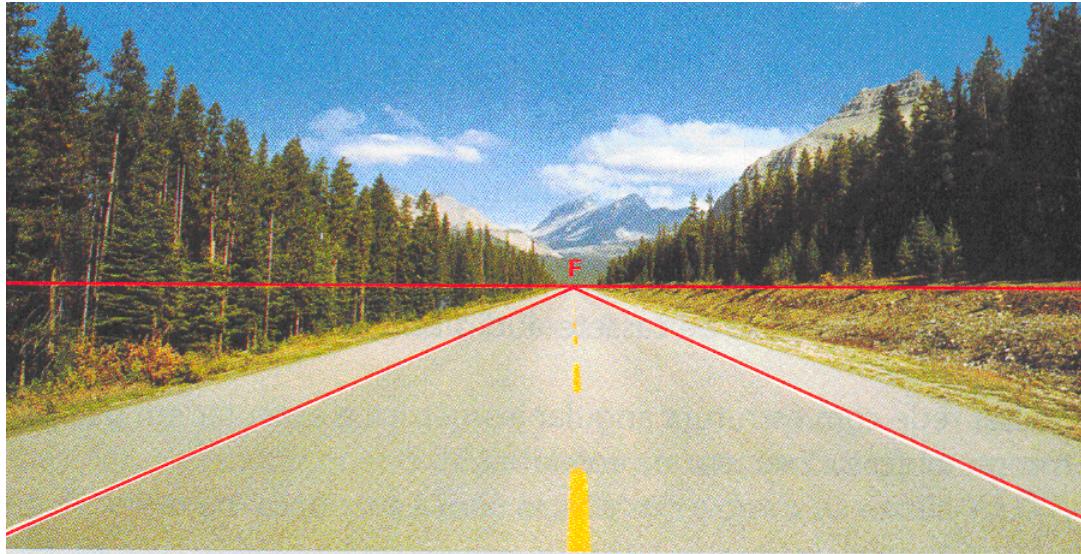
Axonometric Projection in Games



[van Dam]

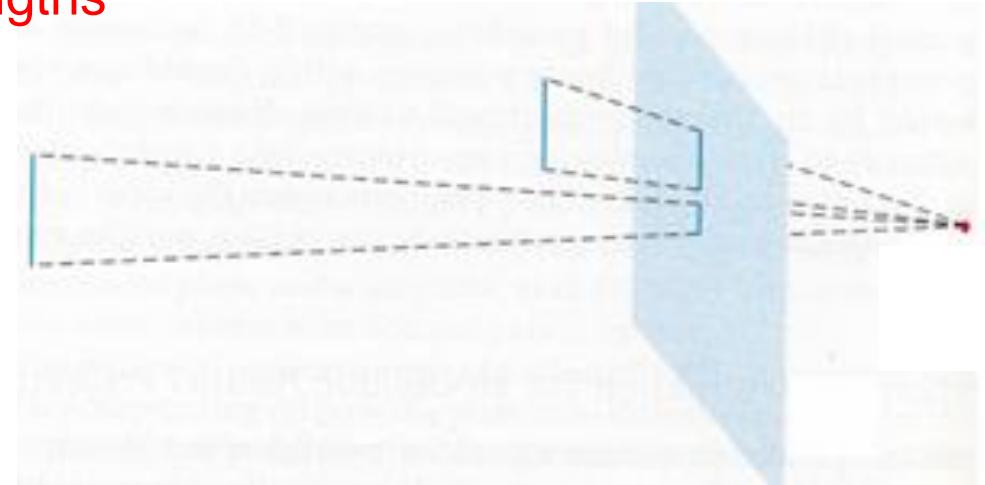
PERSPECTIVE PROJECTIONS

Perspective Projections



Perspective Projection

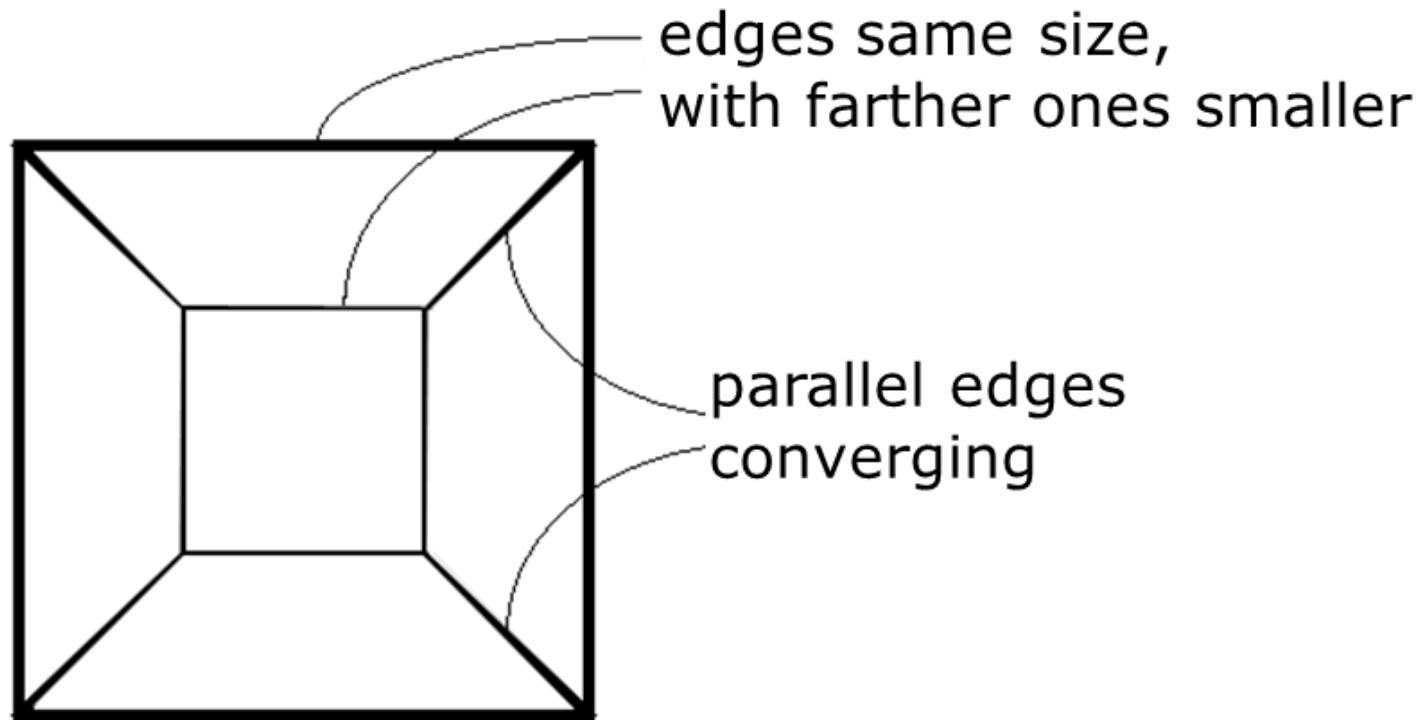
- The projections of straight-line segments with the **same length**, but located at different distances from the projection plane, are projected with **different lengths**



Regarding the parallel projections:

- It generates more realistic images
- But it does not preserve relative sizes of objects
- It requires more calculations

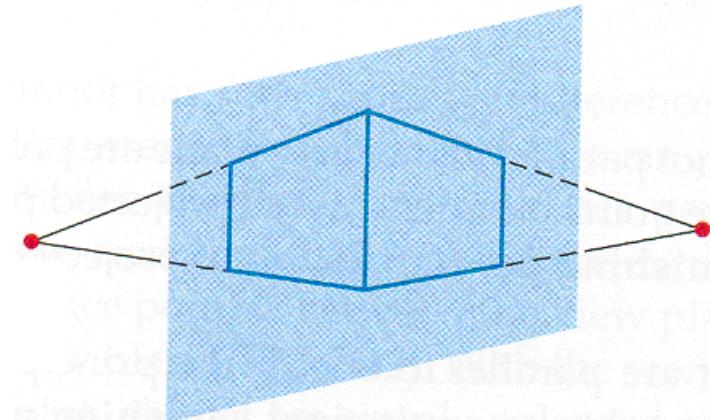
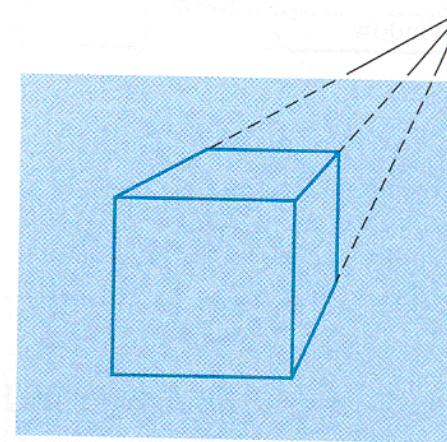
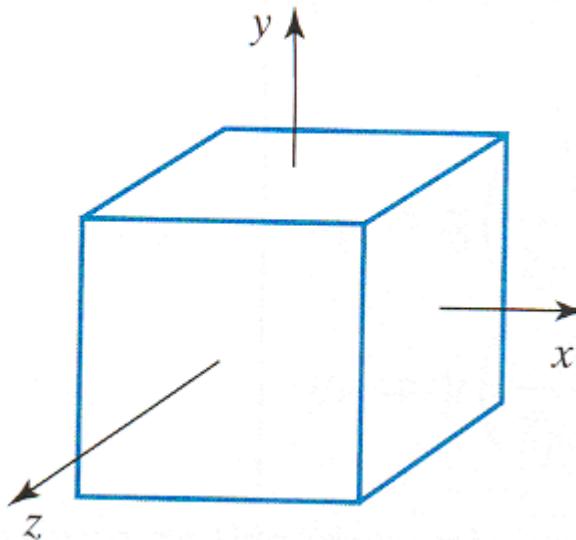
Perspective Projection



[van Dam]

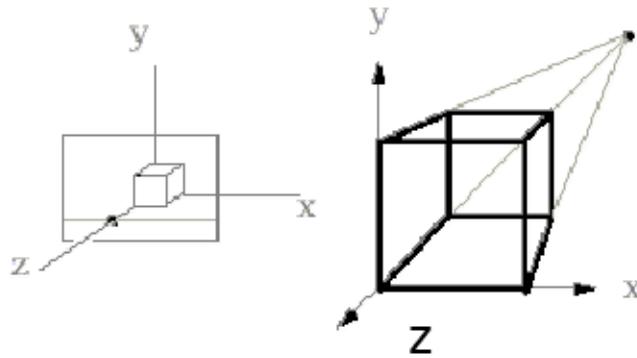
Vanishing Points

- Straight-lines, parallel to a coordinate axis that intersects the projection plane, converge to that axis' vanishing point

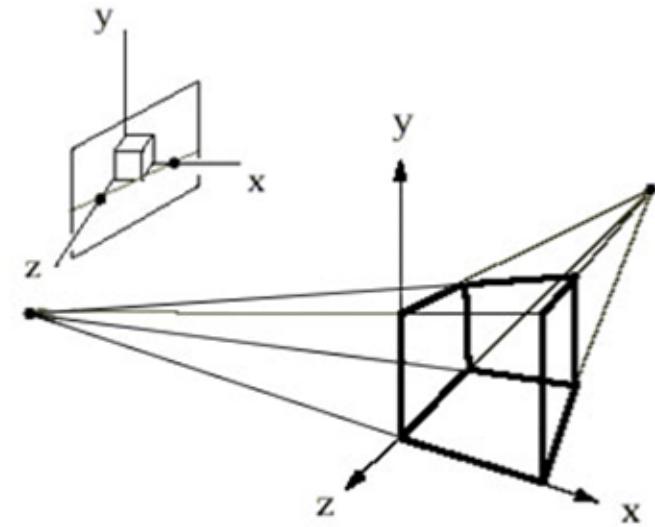


Number of vanishing points:
number of coordinate axes intersecting the projection plane

Vanishing Points



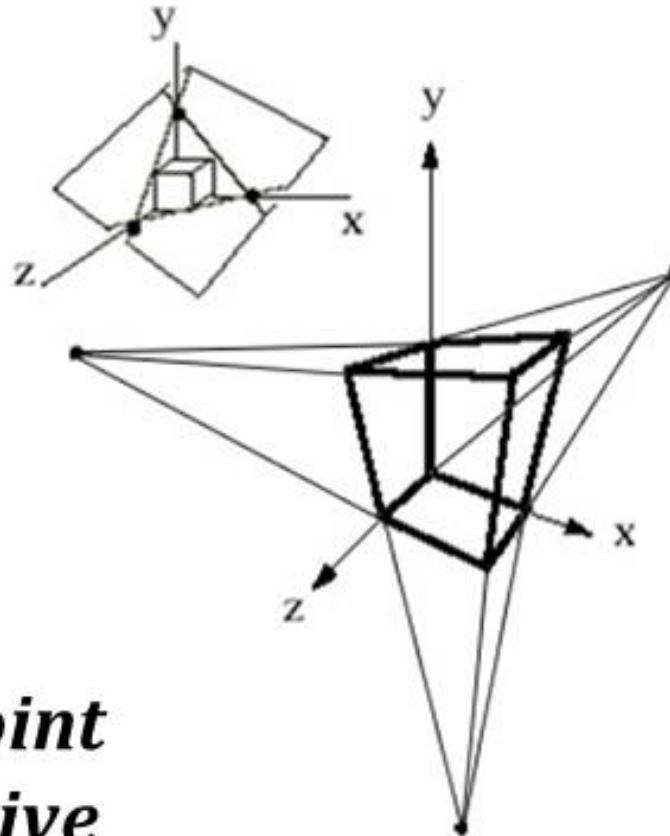
One Point Perspective
(z-axis vanishing point)



Two Point Perspective
(z and x-axis vanishing points)

[van Dam]

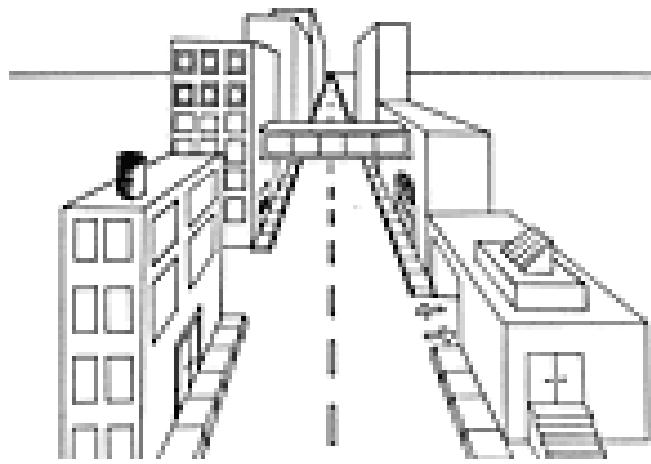
Vanishing Points



***Three Point
Perspective***
(z , x , and y -axis vanishing points)

[van Dam]

1 and 2 vanishing points

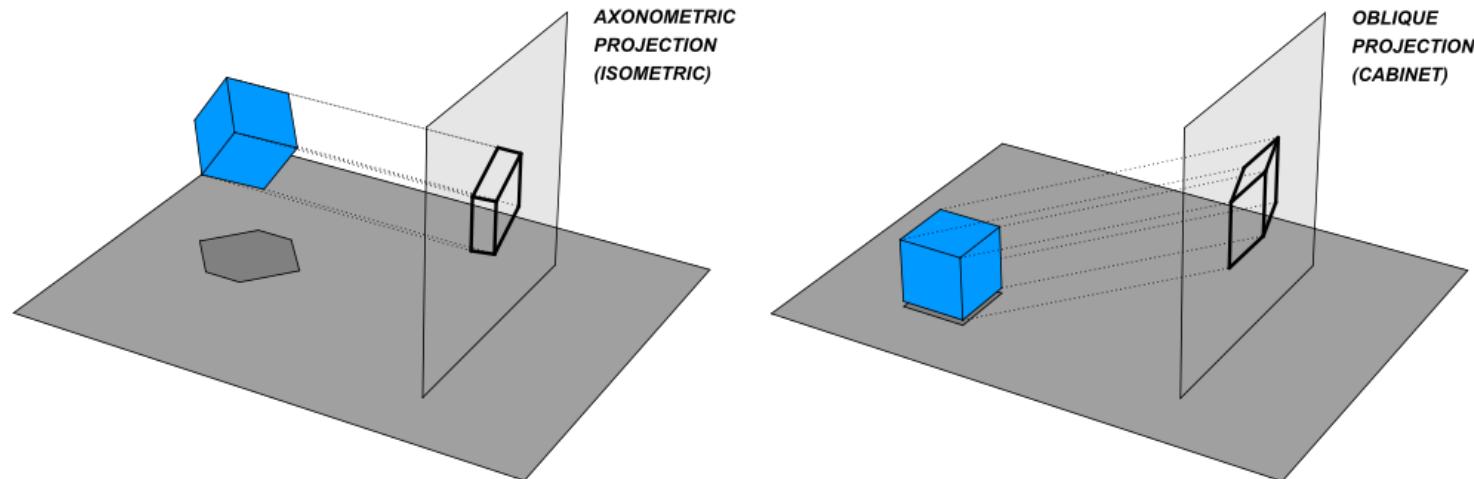
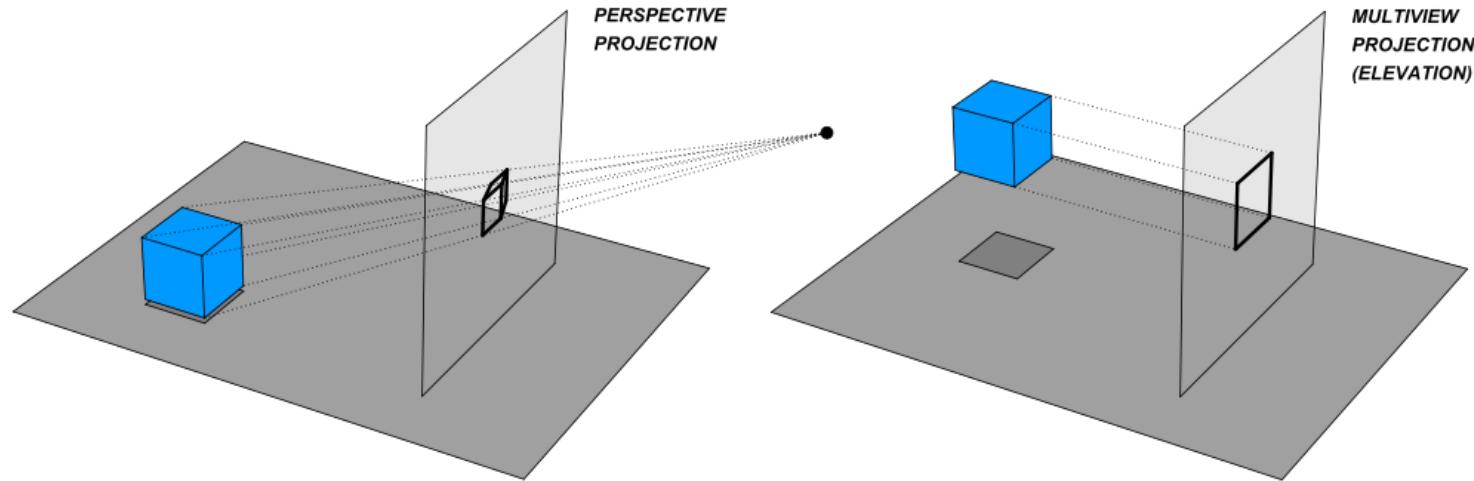


Frontal perspective



Angular perspective

Cube – Various projections



MATRICIAL REPRESENTATION

The Mathematics

- Projection is achieved through matrix **multiplication**, using a **(4 x 4)** projection matrix in **homogeneous coordinates**
- The **projection matrix** can be **concatenated** with the **model-view matrix** to carry out any modeling transformations before the actual projection
 - Animations
 - More complex projections are decomposed into a sequence of simpler transformations
- Let's consider the **simplest cases**, when the **projection plane is XOY or a plane parallel to XOY**

Projection plane at $z = d$

$P(x, y, z)$ – original point

$P_p(x_p, y_p, z_p)$ – projected point

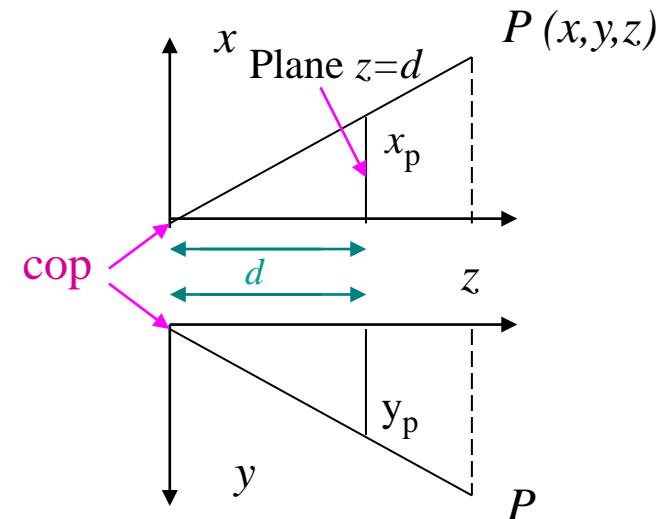
Distance ratios:

$$x_p/d = x/z \quad y_p/d = y/z$$

Multiplying by d :

$$x_p = \frac{d \cdot x}{z} = \frac{x}{z/d}$$

$$y_p = \frac{d \cdot y}{z} = \frac{y}{z/d}$$



Dividing by z implies that objects further away appear smaller

Projection plane at $z = d$

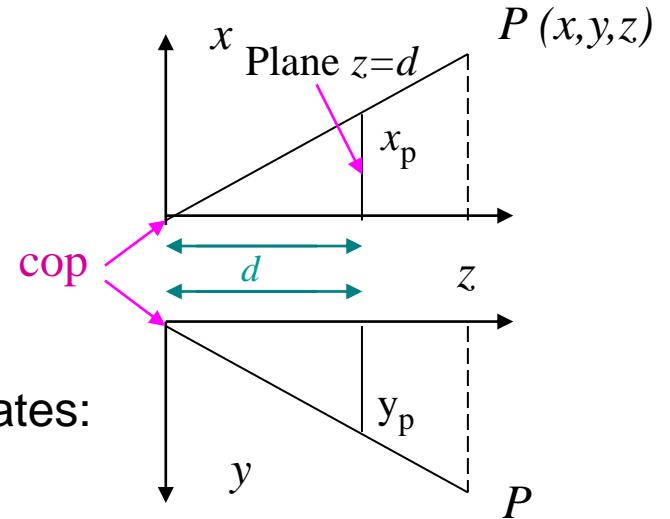
$P(x, y, z)$ – original point

$P_p(x_p, y_p, z_p)$ – projected point

All z values are possible **except** $z=0$

The projection matrix in homogeneous coordinates:

$$M_{pers} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \longrightarrow P_p = M_{pers} \cdot P$$



Center of projection at $z = -d$

$P(x, y, z)$ – original point

$P_p(x_p, y_p, z_p)$ – projected point

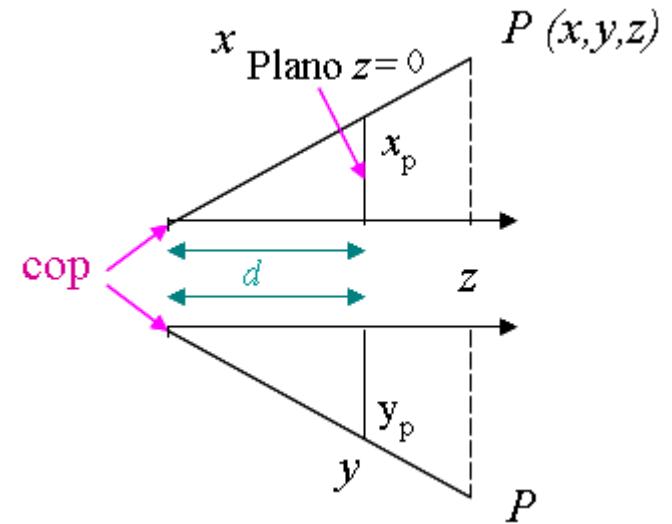
Distance ratios:

$$x_p/d = x/(z + d) \quad y_p/d = y/(z + d)$$

Multiplying by d :

$$x_p = \frac{d \cdot x}{z + d} = \frac{x}{z/d + 1}$$

$$y_p = \frac{d \cdot y}{z + d} = \frac{y}{z/d + 1}$$



$$M'_{pers} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & \textcolor{red}{0} & 0 \\ 0 & 0 & 1/d & 1 \end{bmatrix}$$

Center of projection at $+\infty$

This matricial representation allows to
replace d with ∞ , and we obtain the
matrix for the orthogonal, parallel
projection on the projection plane $z=0$:

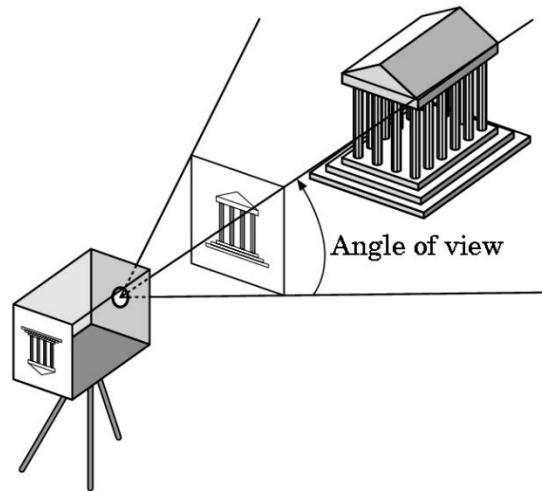
$$M_{ortho} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

What are the coordinates of a projected point ?
Is that the expected result ?

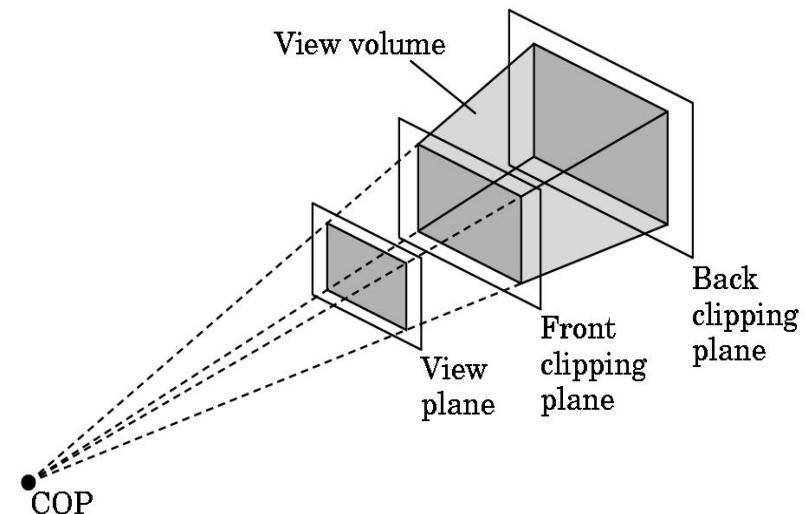
VIEW VOLUME & CLIPPING

Clipping

- The virtual camera only “sees” part of the world or object space
 - View volume

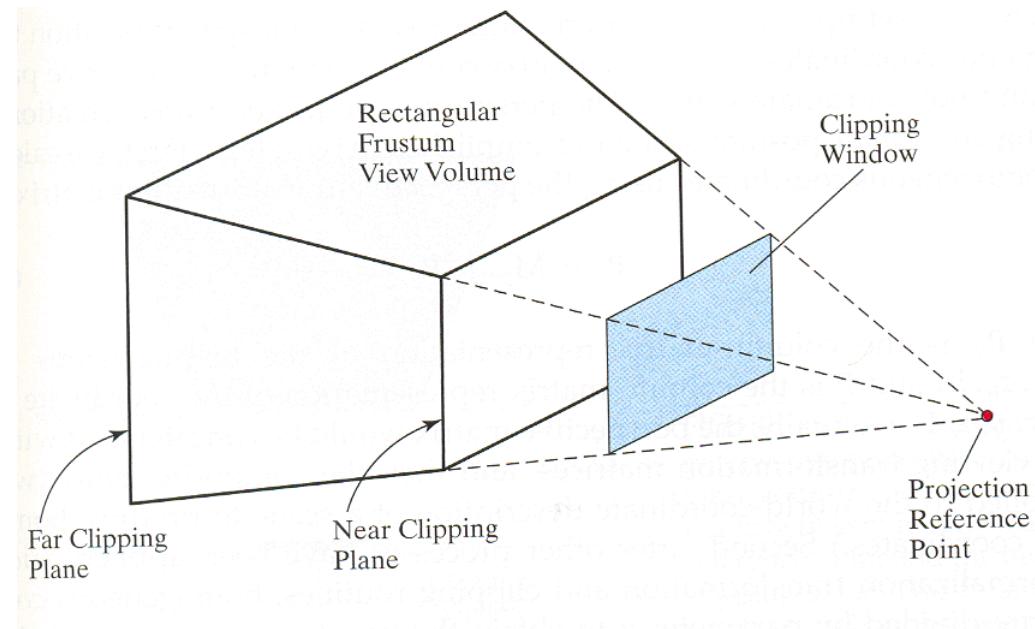
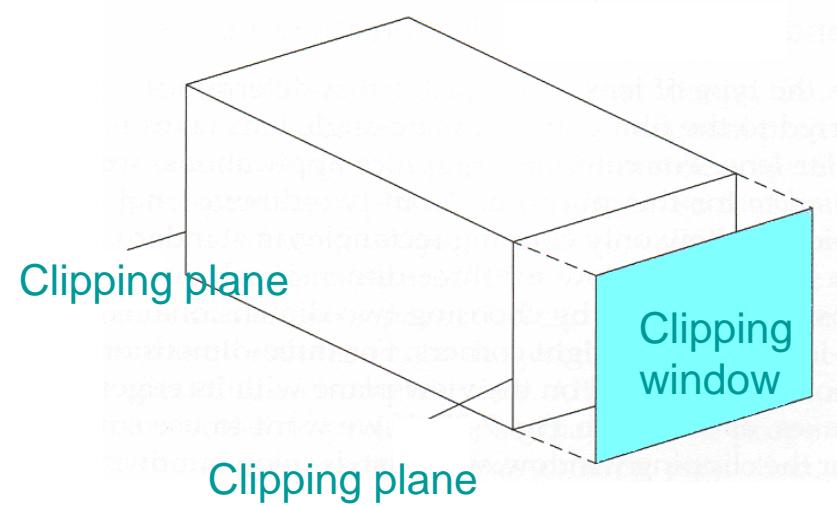


[Angel]

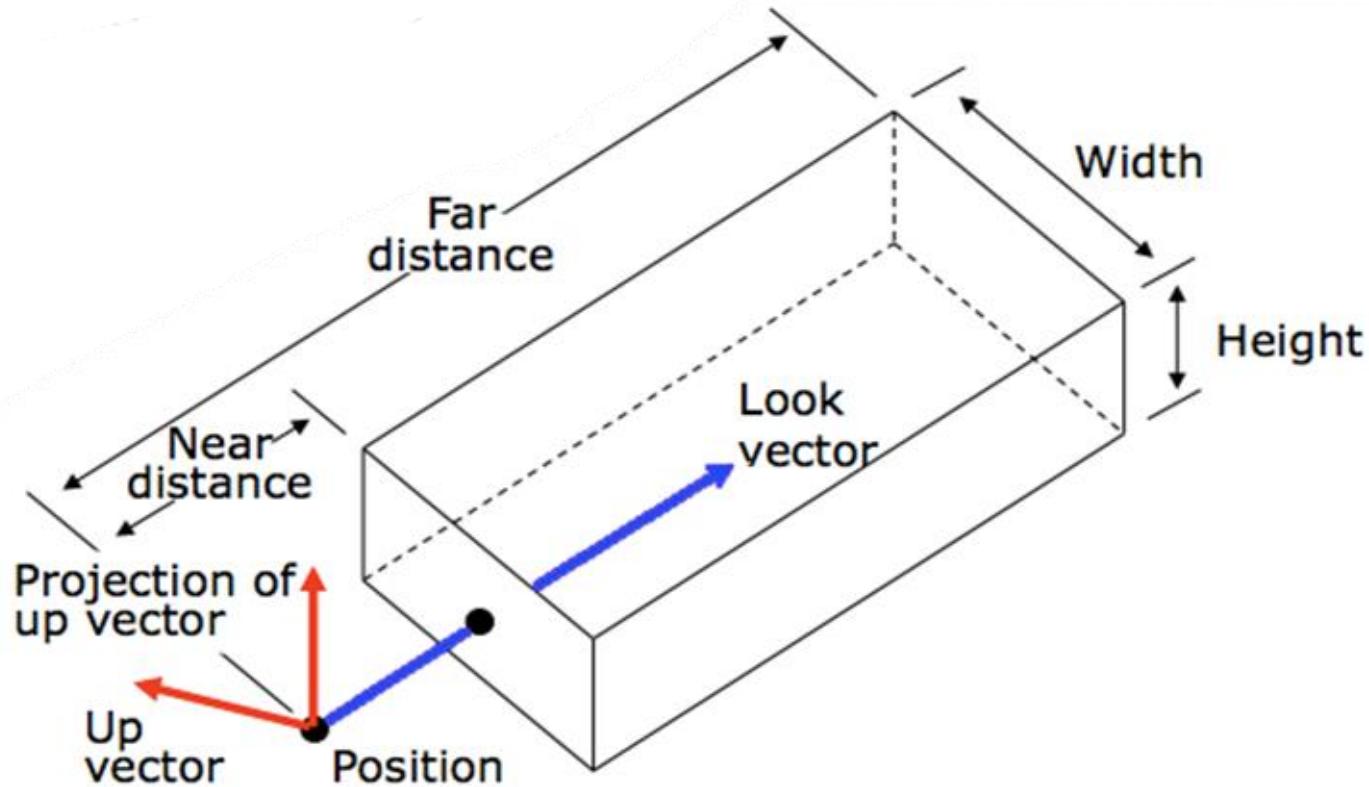


How to limit what is observed ?

- Clipping window on the projection plane
- View volume in 3D

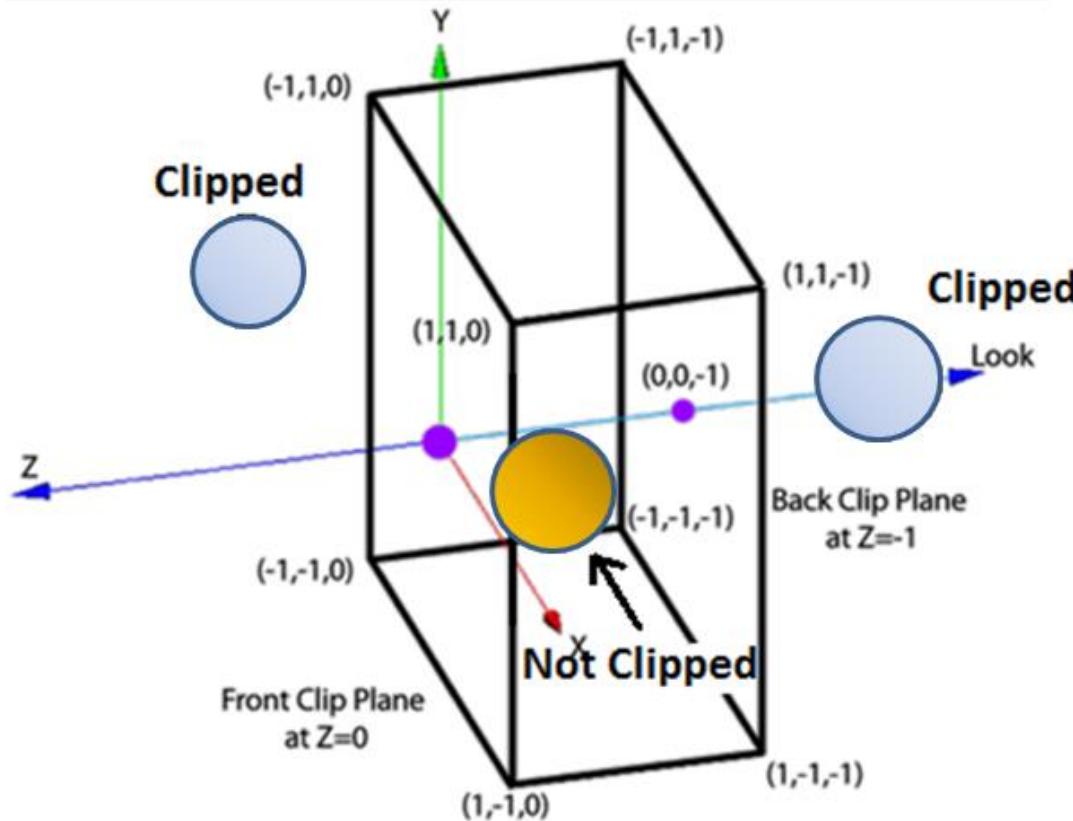


Parallel Projection – View Volume



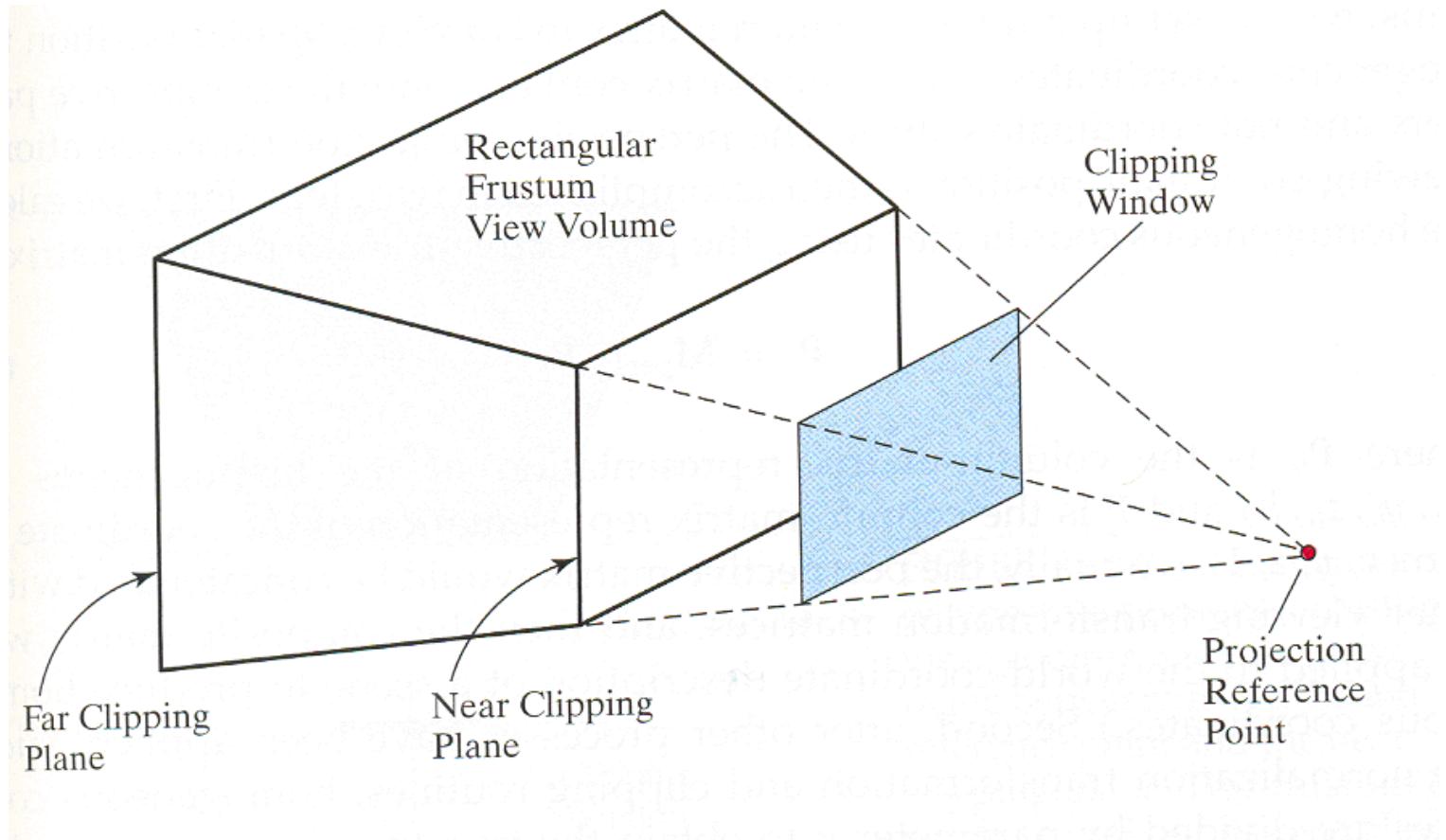
[van Dam]

Clipping against the View Volume

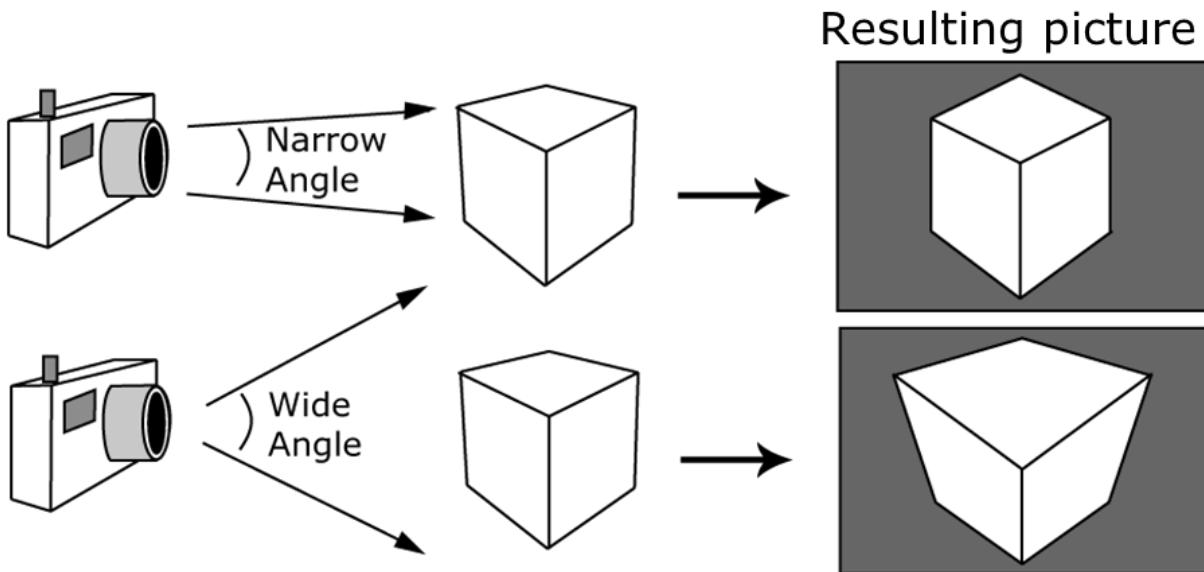
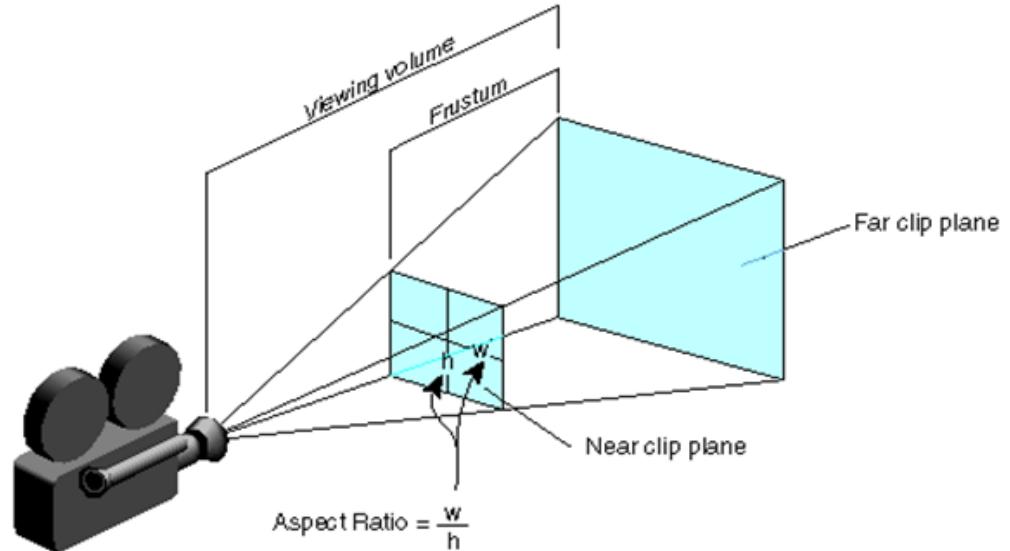


[van Dam]

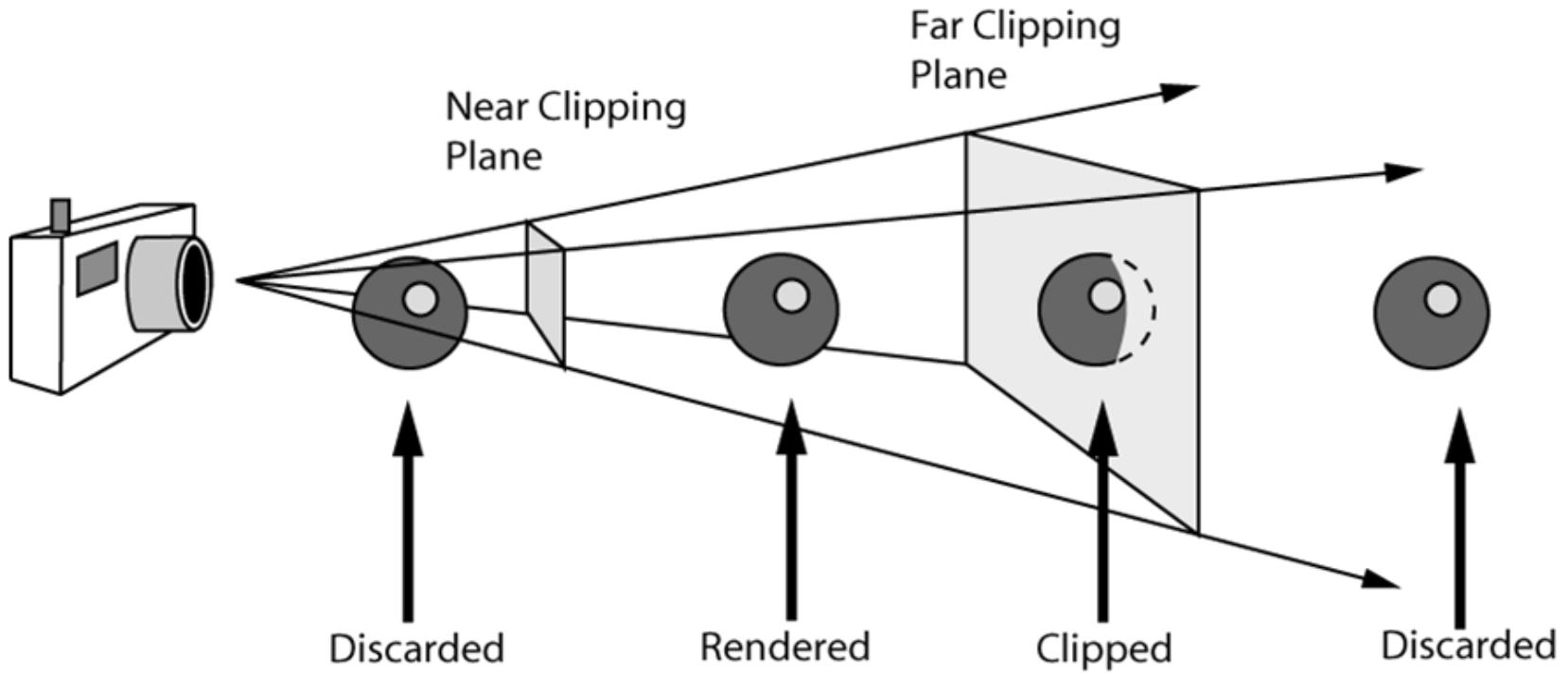
Perspective Projection – View Volume



View Angle



Clipping Planes

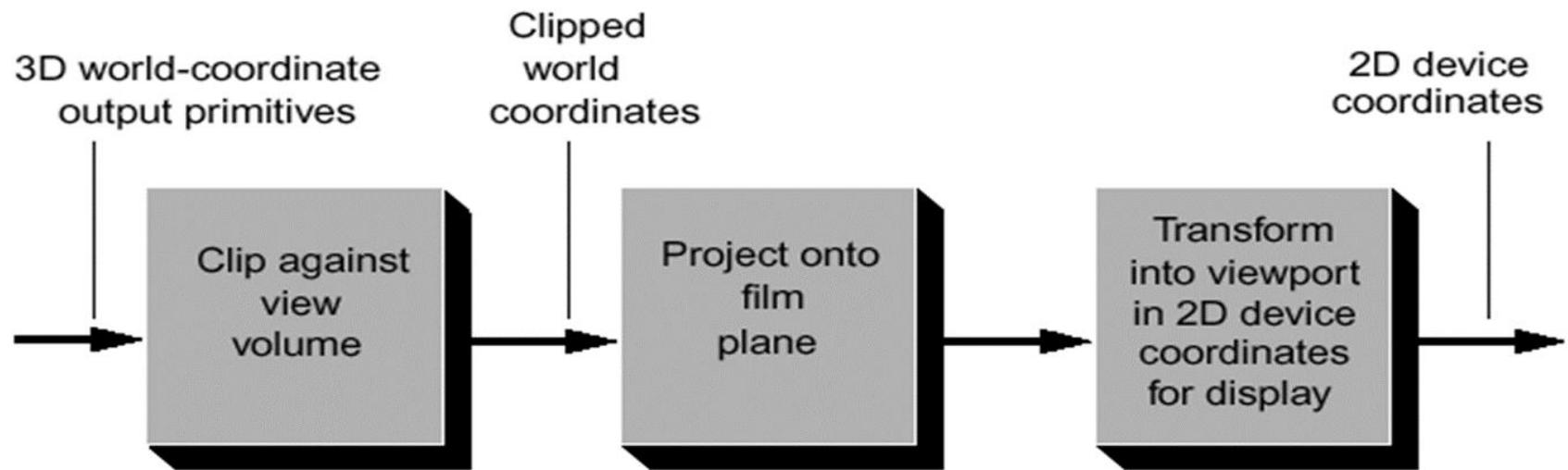


[van Dam]

3D Viewing

- How to view primitives that are **outside** the view volume ?
 - **Translate !**
- How to view a **side face** of a model ?
 - **Rotate !**
- ...

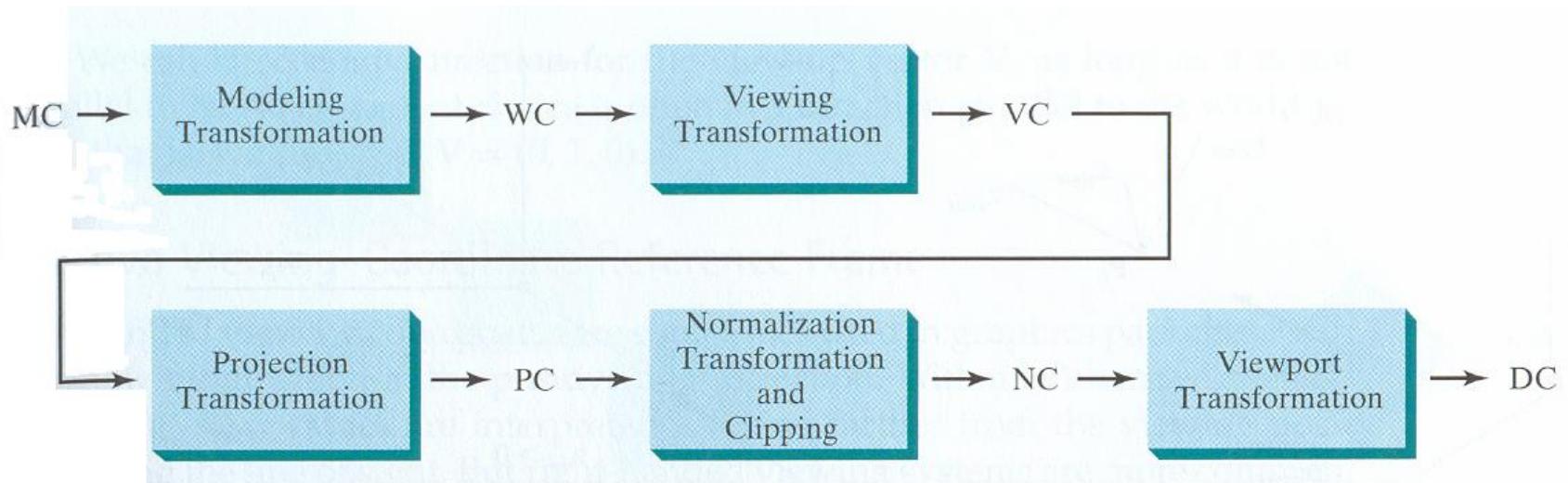
Pipeline



[van Dam]

THE 3D VIEWING PIPELINE

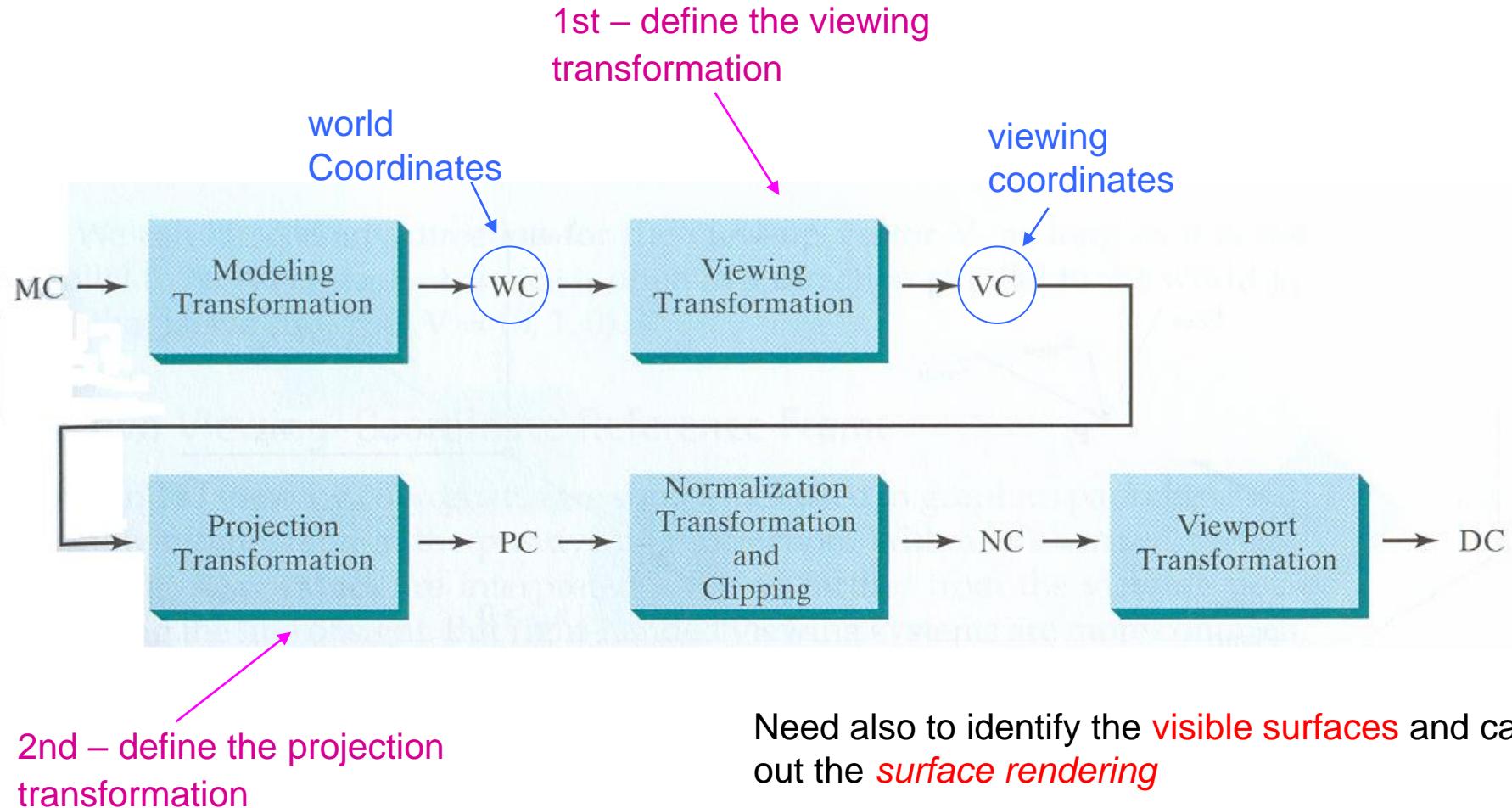
3D visualization pipeline



[Hearn & Baker]

3D visualization pipeline

From **scene** coordinates to **device** coordinates:



3D visualization pipeline

- Instantiate **models**
 - Position, orientation, size
- Establish **viewing parameters**
 - Camera position and orientation
- Compute **illumination** and **shade polygons** ←
- Perform **clipping**
- Project into 2D
- Rasterize

3D visualization pipeline

- Main operations represented as **point transformations**
 - Homogeneous coordinates
 - Transformation matrices
 - Projection matrix
 - Matrix multiplication

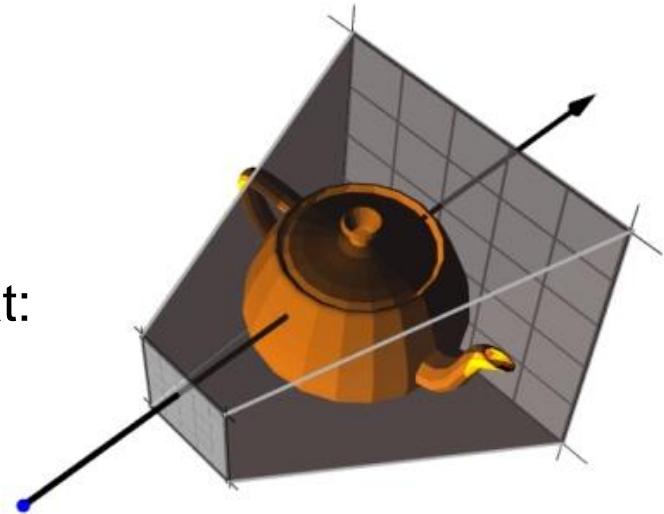
3D visualization pipeline

- Each object is processed **separately**
 - 3D triangles
- Object / triangle inside the **view volume** ?
 - No : go to next object / triangle
- **Rasterization**
 - Compute the location on the screen of each triangle
 - Compute the **color** of each pixel

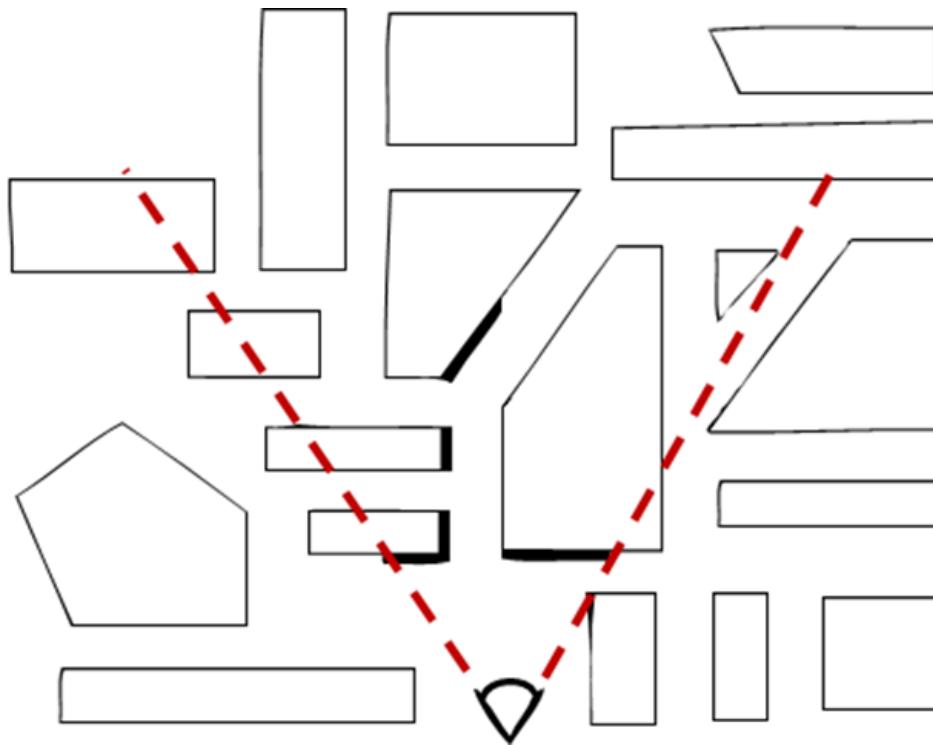
VISIBLE SURFACE DETERMINATION

Visible Surface Determination

- The **visibility** problem
 - Which primitives – after **modeling** transformations, **projection** and **lighting** calculations – contribute for each image pixel
- In general, we solve the dual problem !!
- Which are the **hidden surfaces / faces** that:
 - are **outside of the view volume** ?
 - are **back-faces** in a closed and convex polyhedron ?
 - are **hidden by other faces** closer to the viewpoint / camera ?



Visible Surface Determination



For each object compute:

- The visible edges and surfaces

Why might objects be hidden ?

- Clipping ?
- Occlusion ?

To render or not to render, that is the question...

[Andy Van Dam]

Clipping vs Occlusion

- Clipping against the view volume
 - It is done at object-level !
- Occlusion / Hidden-Surface Removal
 - It is done at scene-level !
 - Compare depth of object / edges / pixels against other objects / edges / pixels

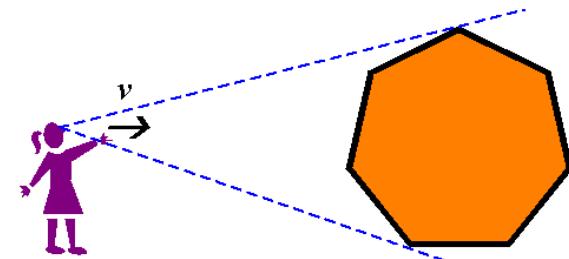
Possible approaches

- Object-precision algorithms
 - Analyze / compare **objects** or parts of objects to determine which surfaces / faces / edges are **fully or partially visible**
 - Back-Face Culling
- Image-precision algorithms
 - Determine visibility for every **pixel** in the **viewing plane**
 - Work in **3D** to get / compare **depth** values (i.e., **z values**)
 - Z-buffer

BACK-FACE CULLING

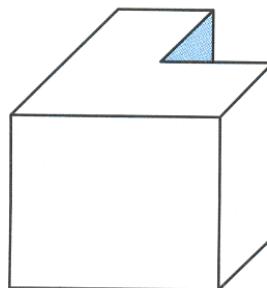
Back-Face Culling

- Sufficient for **a single convex polyhedron** which is not sectioned by clipping

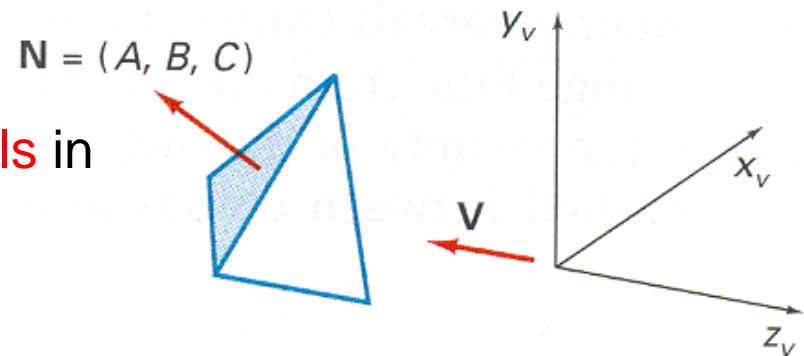


- Not sufficient for**

- concave polyhedra
- when there are **two or more models** in front of each other



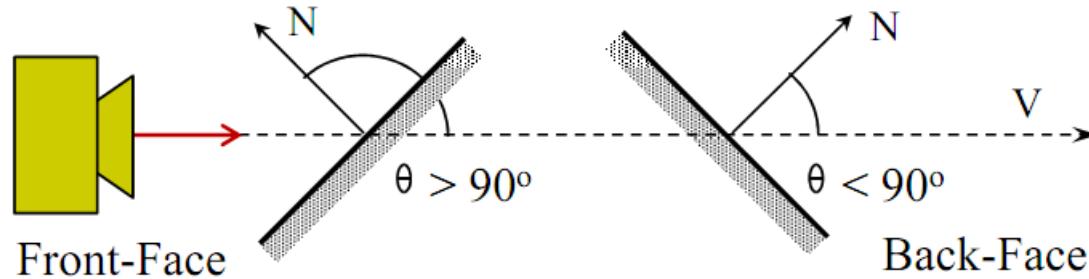
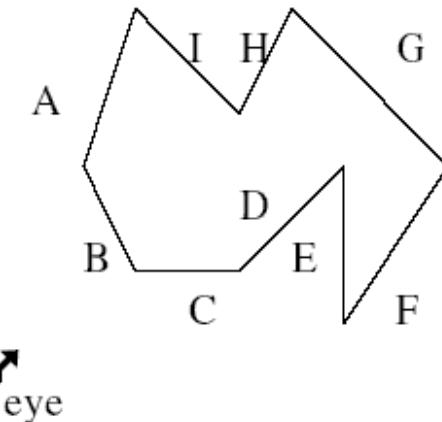
Concave model with a **partially visible face**



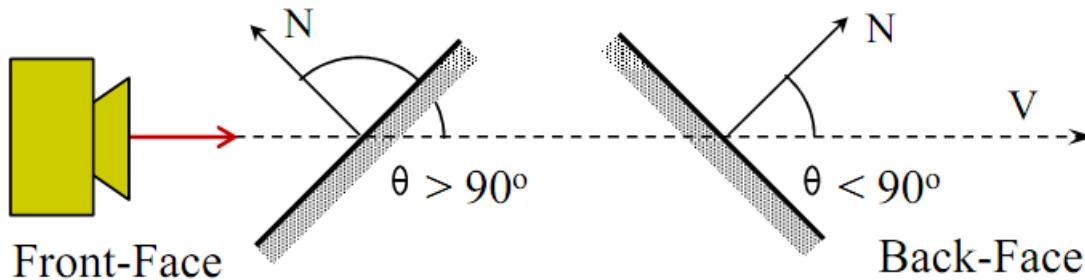
When looking at the negative $z z'$ semi-axis, a polygon is a **back-face** if $C \leq 0$

Back-Face Culling

- What to do in a general case ?
- For each face, compute the angle between
 - The **normal vector** to the face
 - The **viewing direction**, defined by the viewpoint



Back-Face Culling

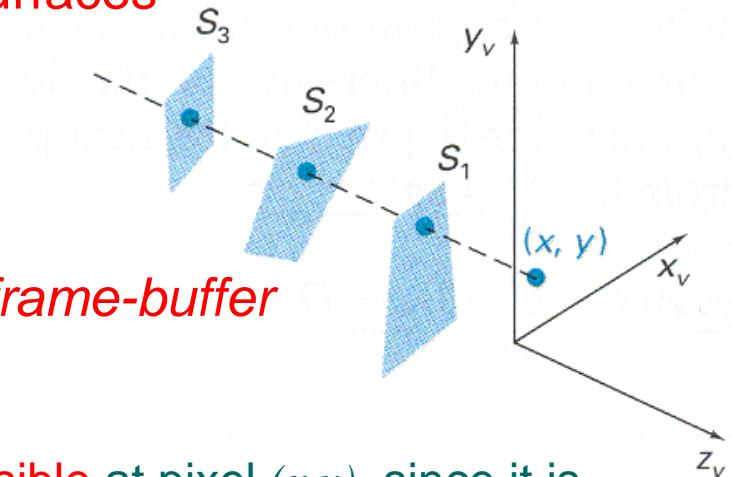


- Efficiency: just compute the **scalar product** !!
 - Reject a face if $\mathbf{N} \cdot \mathbf{V} > 0$
 - What happens if $\mathbf{N} \cdot \mathbf{V} = 0$?
- Simplification when $\mathbf{V} = (0, 0, -1)$!!
- On average, approx. **half** of the faces are removed!

Z-BUFFER

Depth-Buffer (z-buffer)

- Works in *image-space*
- Compares the **depth** of each surface relative to each **pixel** in the viewplane
- Fast and easy to implement for **planar surfaces**
- Can be adapted for curved surfaces
- Needs a **depth-buffer** in addition to the **frame-buffer**



S1 is **visible** at pixel (x,y) , since it is closer to the viewplane

Z-Buffer Algorithm

- Draw **every polygon** that can't be rejected trivially
 - I.e., it is totally outside the view volume
- If a piece (one or more pixels) of a polygon that is **closer to the front** is found
- **Paint over whatever was behind it**
- Use plane equation for polygon, $z = f(x, y)$
- Note: use **positive z** here [0, 1]

Z-Buffer Algorithm

```
void zBuffer() {  
    int x, y;  
    for (y = 0; y < YMAX; y++)  
        for (x = 0; x < XMAX; x++) {  
            WritePixel (x, y, BACKGROUND_VALUE);  
            WriteZ (x, y, 1);  
        }  
    for each polygon {  
        for each pixel in polygon's projection {  
            //plane equation  
            double pz = Z-value at pixel (x, y);  
            if (pz <= ReadZ (x, y)) {  
                // New point is closer to front of view  
                WritePixel (x, y, color at pixel (x, y))  
                WriteZ (x, y, pz);  
            }  
        }  
    }  
}
```

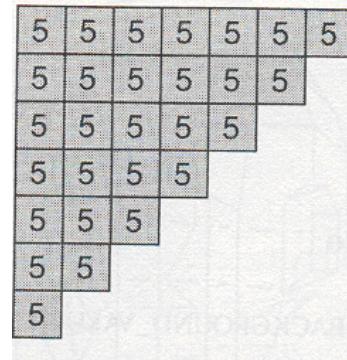
[Andy Van Dam]

Example

Initial *z-buffer* values

0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0

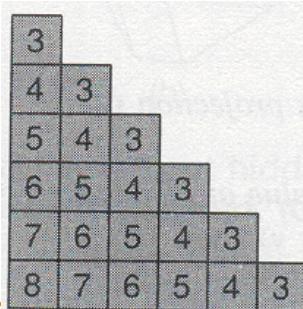
Depth-values for polygon



Overwrite *z-buffer*

5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0

Another polygon



Final *z-buffer*

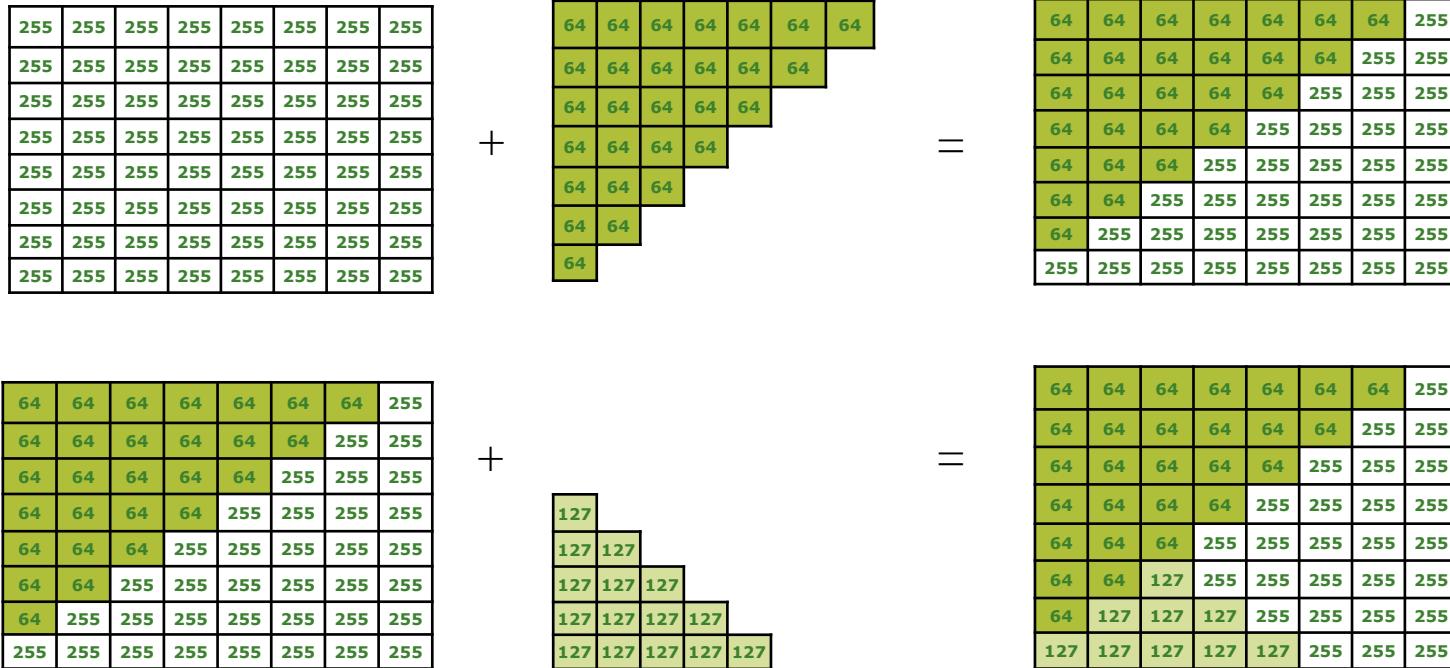
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0
5	5	5	5	5	5	5	5	0

Note:

0 – background depth

Max – viewplane

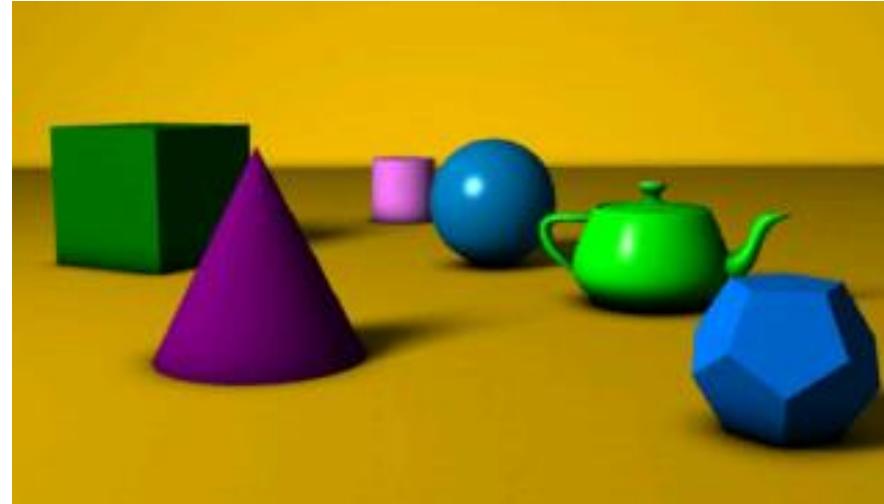
Another example



integer Z-buffer with
near = 0, far = 255

[Andy Van Dam]

3D scene



Z-buffer



Z-Buffer Algorithm

- Advantages:

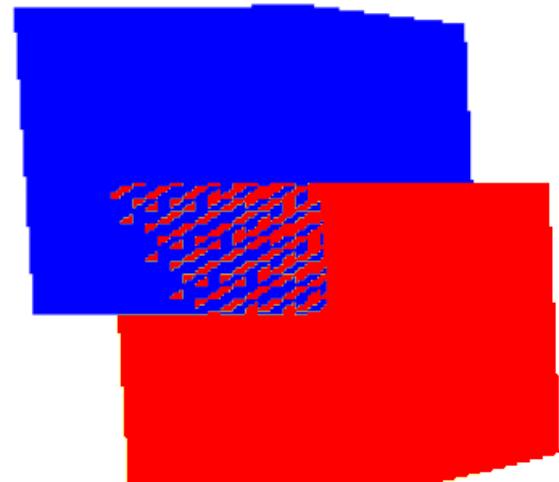
- Easy, no need to previously sort the various surfaces
- Fast

- Disadvantages:

- Need for additional memory
- Depth precision problems / limitations
- It finds one visible surface for each pixel
- I.e., it can only handle opaque surfaces

Z-Fighting

- Z-fighting occurs when two **primitives have similar values** in the z-buffer
 - Coplanar polygons (two polygons that occupy the same space)
 - One is arbitrarily chosen over the other, but z varies across the polygons and binning will cause artifacts
 - Behavior is deterministic: the same camera position gives the same z-fighting pattern



Two intersecting cubes

[Andy Van Dam]

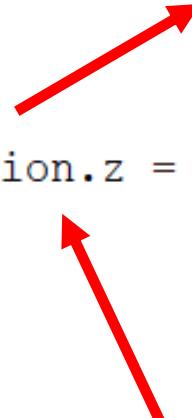
Possible References

- Projections and the Viewing Pipeline are presented in any Computer Graphics book
- E. Angel and D. Shreiner. Interactive Computer Graphics, 7th Ed., Addison-Wesley, 2015
- J M Pereira, et al. Introdução à Computação Gráfica. FCA, 2018

THREE.JS – THE CAMERA

Three.js – PerspectiveCamera

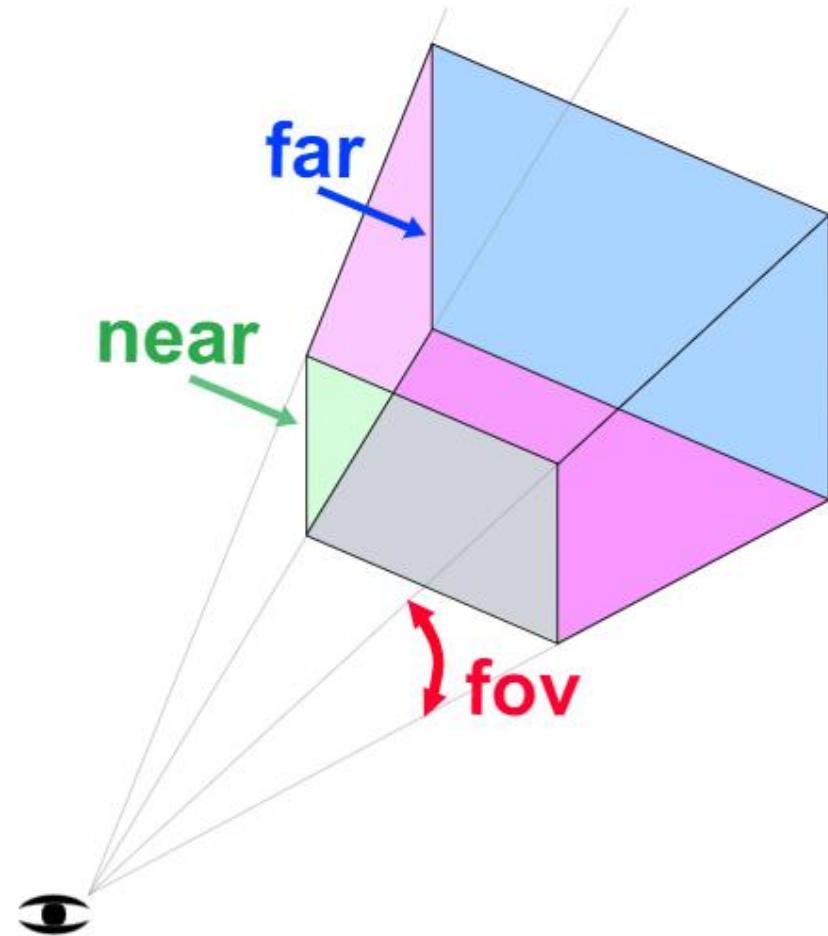
```
// The CAMERA  
  
// --- Where the viewer is and how he is looking at the scene  
  
camera = new THREE.PerspectiveCamera( 70,  
    window.innerWidth / window.innerHeight, 1, 1000 );  
  
camera.position.z = 400;
```

Two red arrows point to the camera parameters in the code. One arrow points to the field of view parameter '70' in the constructor of the PerspectiveCamera. Another arrow points to the z-position parameter '400' in the assignment statement.

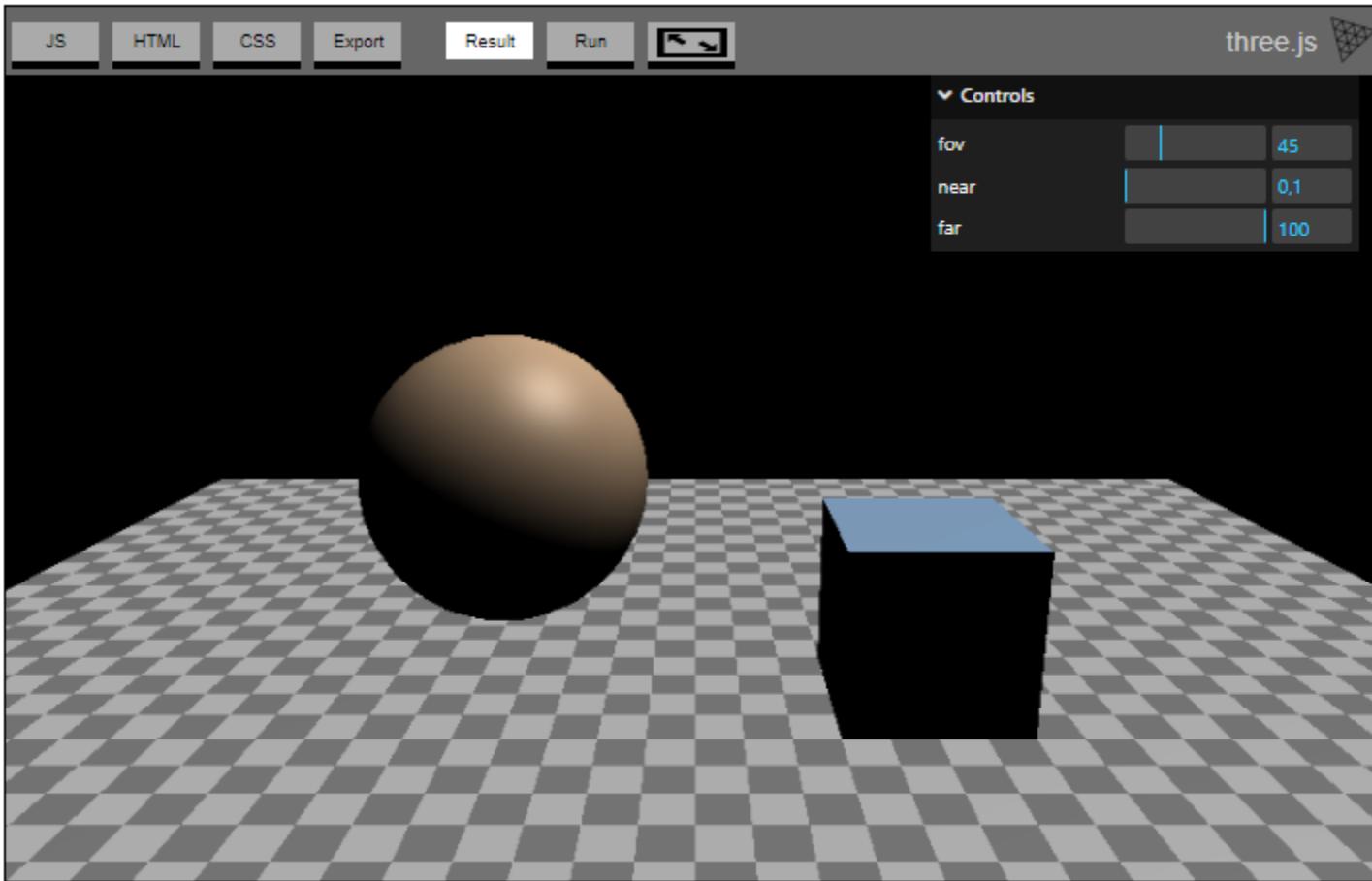
Three.js – PerspectiveCamera

- The most common camera in three.js
- Originates a 3D view where things in the distance appear smaller than things up close
- Defines a **frustum** as **perspective view-volume**

Three.js – Prespective frustum

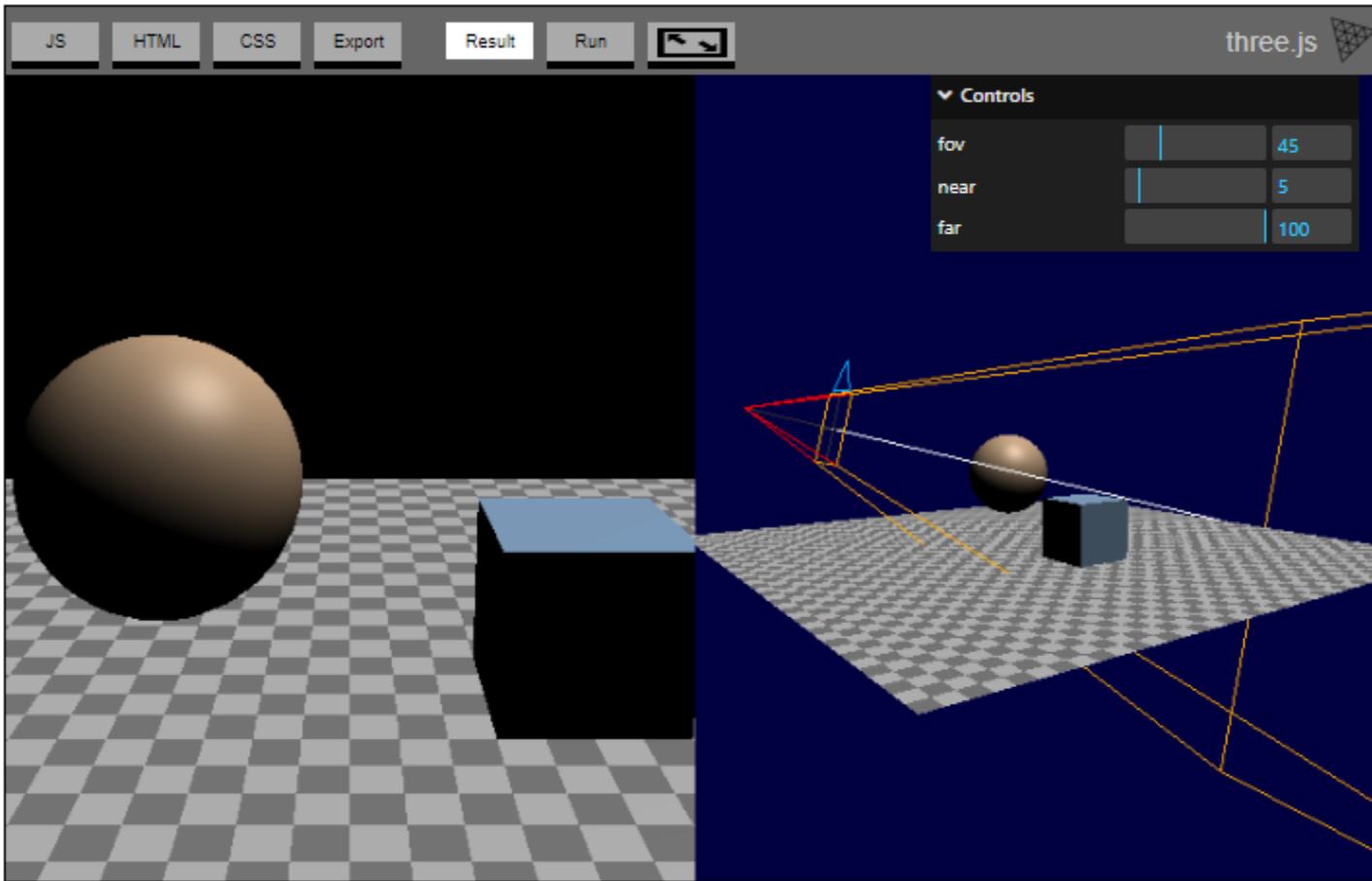


Three.js – PerspectiveCamera



<https://threejs.org/manual/examples/cameras-perspective.html>

Three.js – PerspectiveCamera



<https://threejs.org/manual/examples/cameras-perspective-2-scenes.html>

Three.js – OrthographicCamera

Cameras

ArrayCamera

Camera

CubeCamera

OrthographicCamera

PerspectiveCamera

StereoCamera

Constants

Constructor

`OrthographicCamera(left : Number, right : Number, top : Number, bottom : Number, near : Number, far : Number)`

`left` — Camera frustum left plane.

`right` — Camera frustum right plane.

`top` — Camera frustum top plane.

`bottom` — Camera frustum bottom plane.

`near` — Camera frustum near plane.

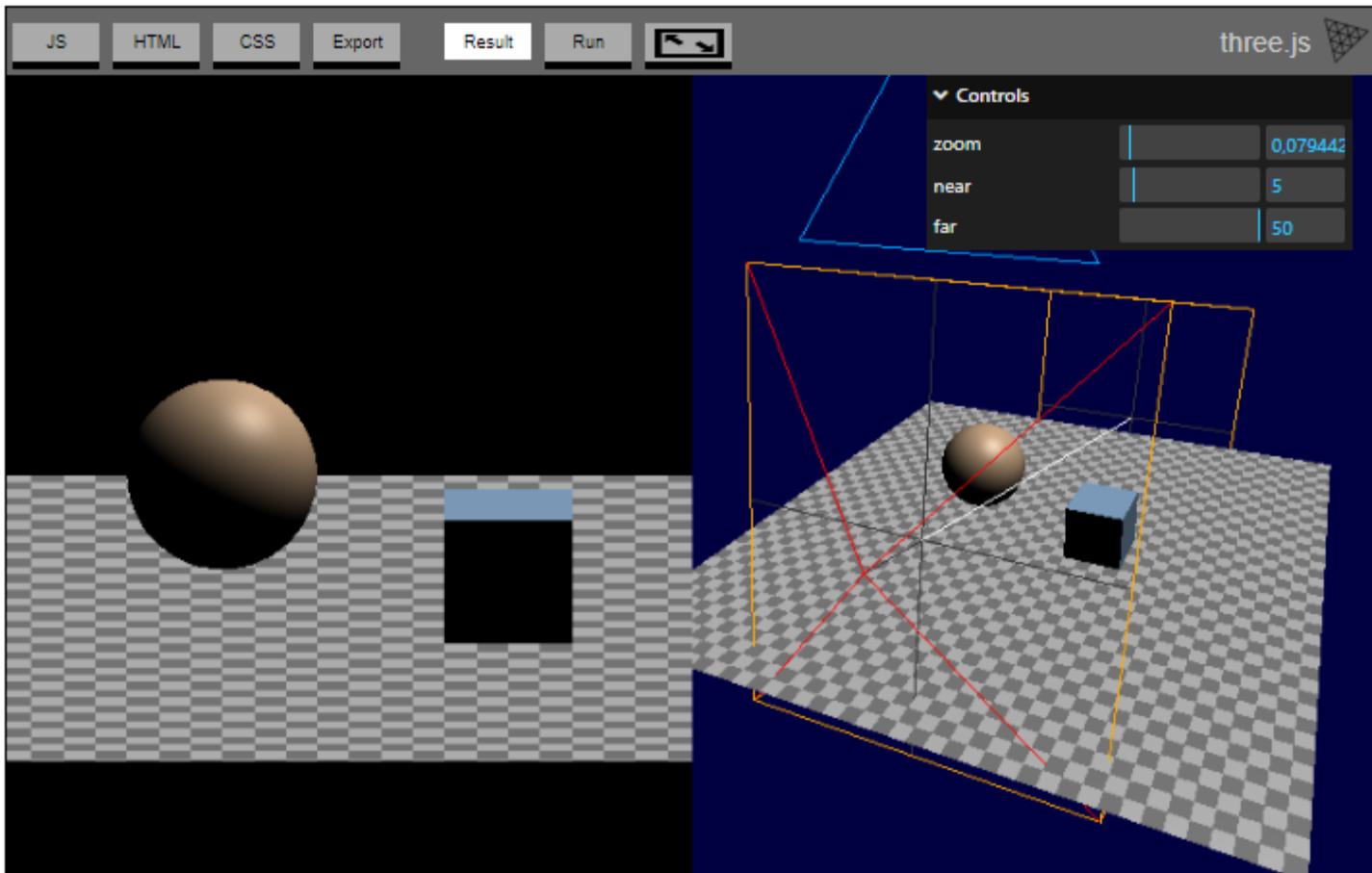
`far` — Camera frustum far plane.

[<https://threejs.org/docs/index.html#api/en/cameras/OrthographicCamera>]

Three.js – OrthographicCamera

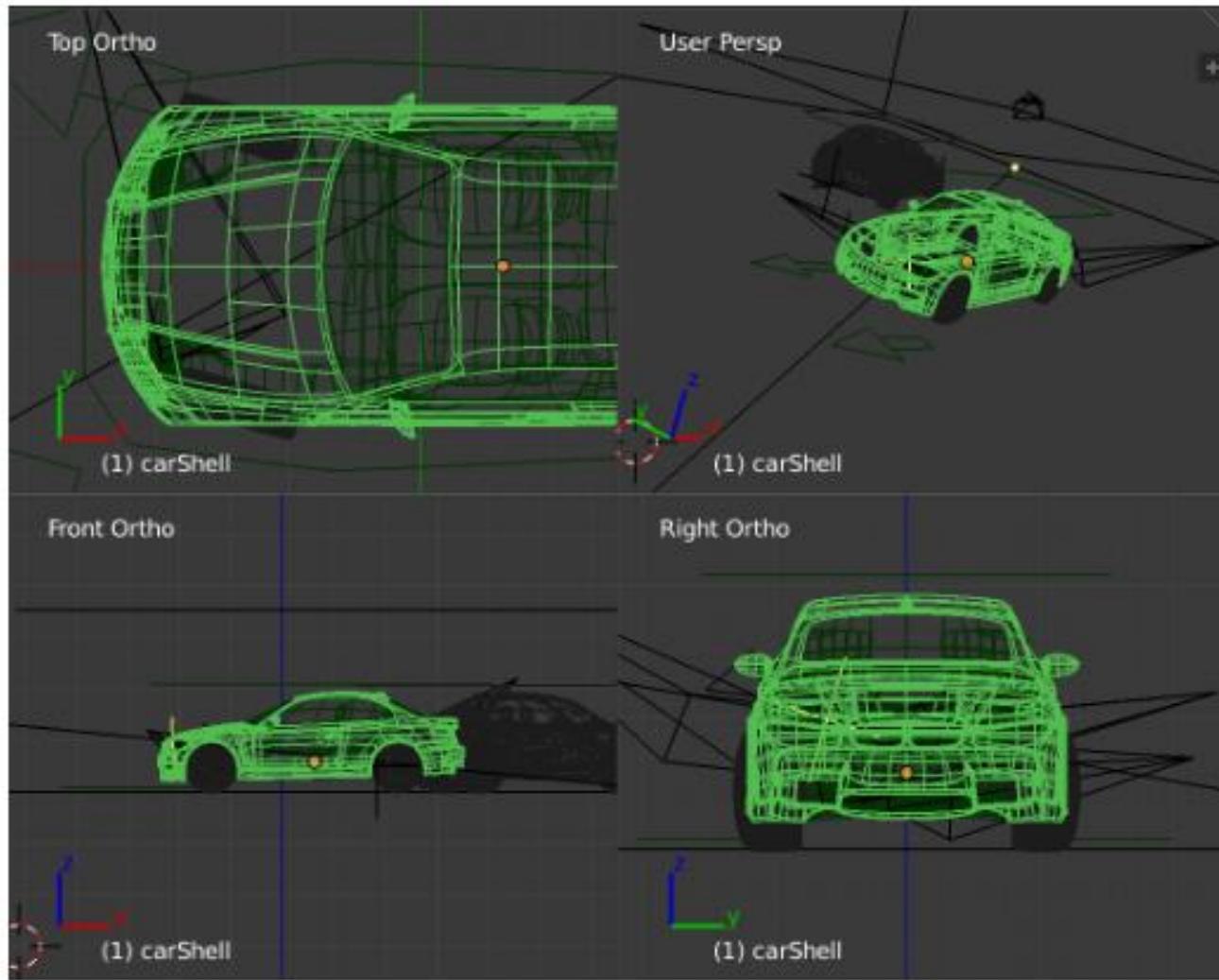
- Originates a 3D view using the orthogonal parallel projection
- Defines a **box** as **orthographic view-volume**
- Useful for **2D applications**
- And for traditional **modeling / design / engineering** scenarios

Three.js – OrthographicCamera



<https://threejs.org/manual/examples/cameras-orthographic-2-scenes.html>

Three.js – OrthographicCamera



ACKNOWLEDGMENTS

Acknowledgments

- Some ideas and figures have been taken from slides of other CG courses.
- In particular, from the slides made available by Beatriz Sousa Santos, Ed Angel and Andy van Dam.
- Thanks!

Acknowledgment

- Example code and figures taken from
<https://threejs.org/manual/#en/cameras>

Illumination and Shading

Joaquim Madeira

April 2022

Topics

- Motivation
- Phong's Illumination model
- Shading Methods
- Three.js – Light Sources & Shading

MOTIVATION

Modeling vs Rendering

■ Modeling

- Create models
- Apply materials to models
- Place models around scene
- Place lights in the scene
- Place the camera



[YouTube Demo](#)

■ Rendering

- Take picture with the camera

[van Dam]

3D visualization pipeline

- Create a **scene** and instantiate **models**
 - Position, orientation, size
- Establish **viewing parameters**
 - Camera position and orientation
- Perform **clipping**
- Compute **illumination** and **shade polygons**
- Project into 2D
- Rasterize



3D visualization pipeline

- Each object is processed **separately**
 - 3D **triangles**
- Object / triangle **inside** the **view volume** ?
 - **No** : go to next object / triangle
- **Rasterization**
 - Compute the location on the screen of each triangle
 - Compute the **color** of each **pixel**

Lighting or Illumination

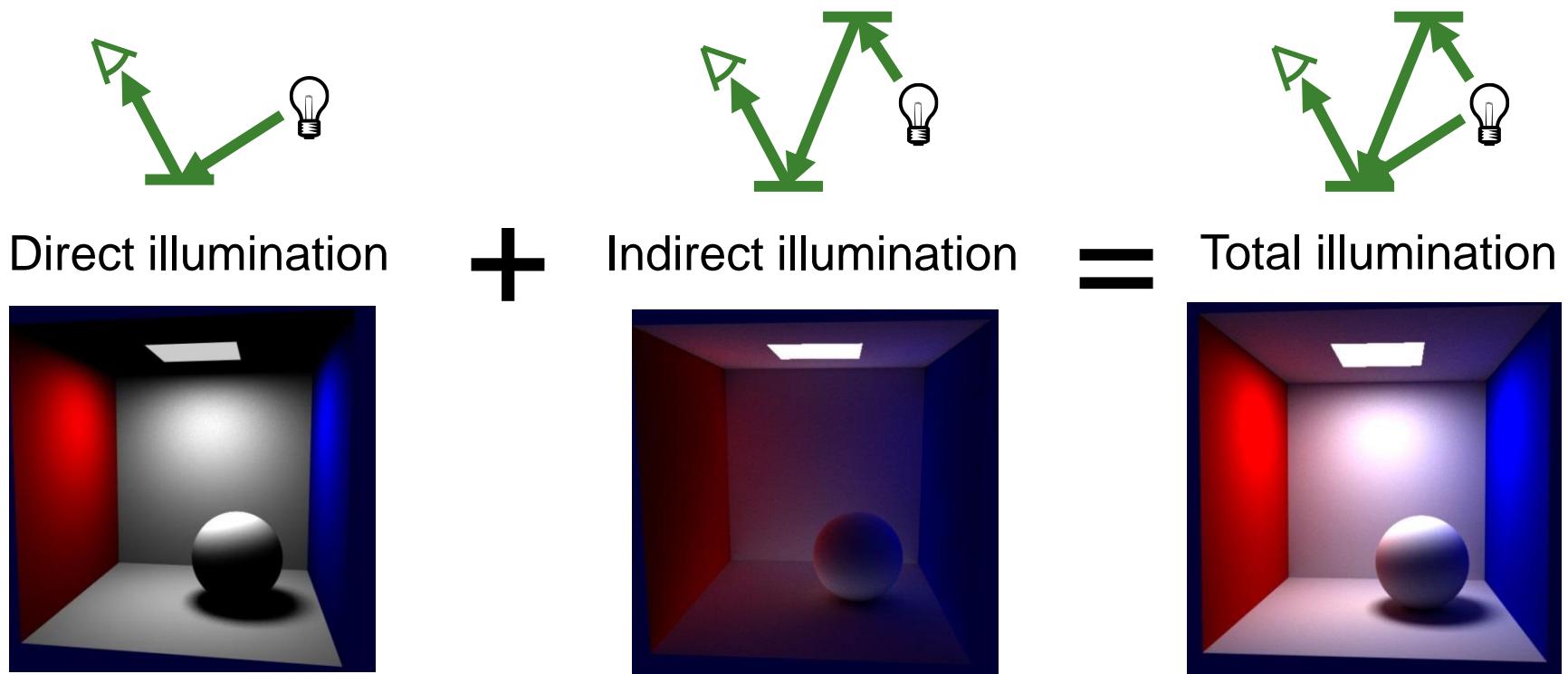
- The process of **computing** the **intensity** and **color** of a sample **point** in a scene as seen by a **viewer**
- It is a function of the **geometry** of scene
 - Models, lights and camera, and their spatial relationships
- And of **material** properties
 - Reflection, absorption, ...

Shading

- The process of *interpolation* of color at points **in-between** those with **known lighting** or illumination
 - Vertices of triangles in a mesh
- Used in many real time graphics applications (e.g., games)
 - Calculating illumination at a point is usually **expensive !**
- **BUT**, in **ray-tracing** only do lighting for samples / pixels
 - More sophisticated / demanding
 - Based on pixels (or sub-pixel samples for super-sampling)
 - **No shading rule**

COMPUTING ILLUMINATION

Global Illumination



[Andy Van Dam]

Local vs Global Illumination



Direct (diffuse + specular) lighting +
indirect specular reflection

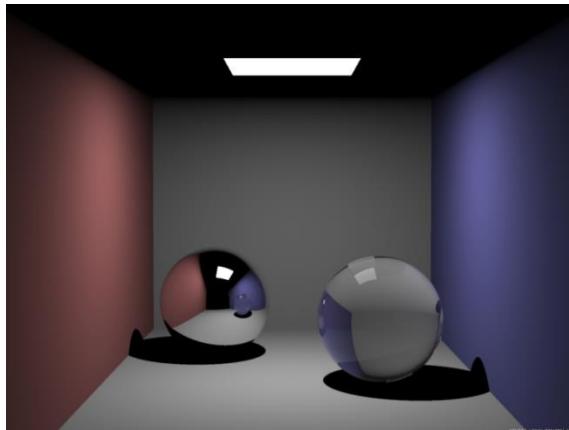


Full global illumination

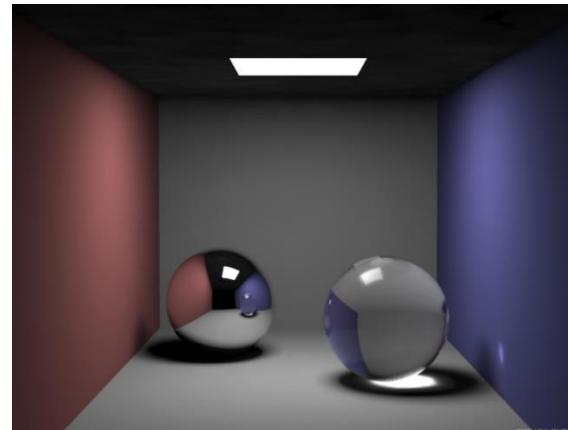
[Andy Van Dam]

Global Illumination – Examples

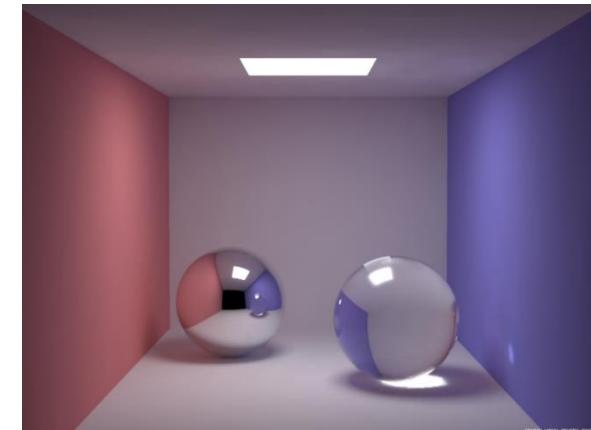
- Take into account **global information** of both **direct** (from emitters) and **indirect illumination** (inter-object reflections)
- Different approximations
 - Advantages and disadvantages; resource requirements
 - More computation gives better results...



Direct illumination + specular reflection
Ray trace



+ soft shadows and caustics
Ray trace + caustic photon map



+ diffuse reflection (color bleeding)
Ray trace + caustic and diffuse photon maps

<http://graphics.ucsd.edu/~henrik/images/global.html>

[Andy Van Dam]

Light Transport Simulation

- Evaluate illumination with enough **samples** to produce final images **without any guessing / shading**
- Often used for high quality renderers, e.g., those used in **FX movies**
 - Can take **days for a single frame**, even on modern render farms
- Some implementations can run in real time on the **GPU**
 - But more complex lighting models that are difficult to parallelize are still run on the **CPU**
- Many simulations use **stochastic sampling**
 - Path tracing, photon mapping, Metropolis light transport

Polygon Rendering

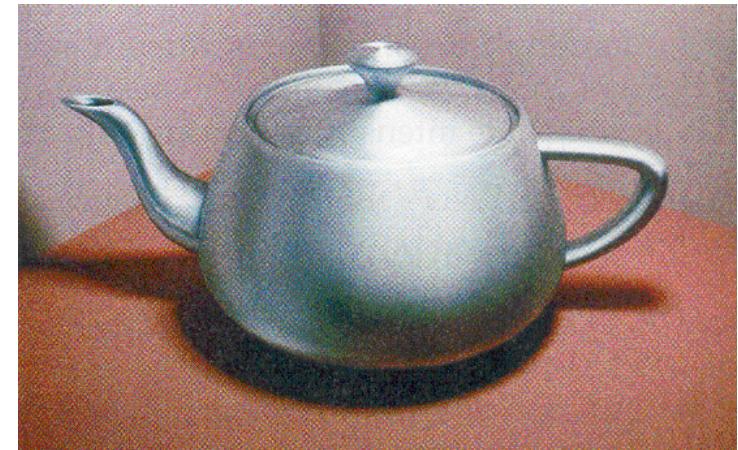
- Evaluate illumination at **several samples**
- **Shade** (using a shading rule) in between to produce **pixels** in the final image
- **Often used** in real-time applications such as computer games
- Done in the **GPU**
- **Lower quality** than light transport simulation !!
 - But **satisfactory** results with various **additions** such as **maps** (bump, displacement, environment)

Polygon Rendering

- Get **realistic images** by :
 - using **perspective projections** of the scene models
 - applying **natural illumination effects** on the visible surfaces
- **Natural illumination effects** are obtained using:
 - an **illumination model** – allows computing the **color** to be assigned to each visible **surface point**
 - a **surface-rendering method** that applies an illumination model and assigns a **color to every pixel**

Polygon Rendering

- Photorealistic images require:
 - a precise representation of the properties of each surface
 - a good description of the scene's illumination
- And might imply modeling:
 - surface texture
 - transparency
 - reflections
 - shadows
 - etc.



Polygon Rendering

- Compute **surface color** based on
 - Type and number of **light sources**
 - **Illumination model**
 - Phong: ambient + diffuse + specular components
 - **Reflective surface properties**
 - Atmospheric effects
 - Fog, smoke
- **Polygons** making up a model surface **are shaded**
 - Realistic representation

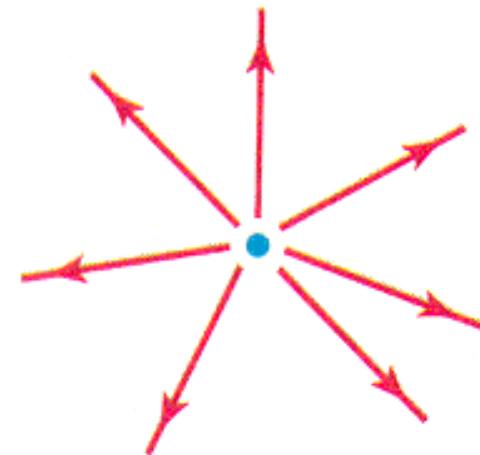
Polygon Rendering

- Illumination models used in Computer Graphics
 - are often an approximation to the Laws of Physics
 - that describe the interaction light-surface
- There are different types of illumination models
 - simple models, based on simple photometric computations (to reduce the computational cost)
 - more sophisticated models, based on the propagation of radiant energy (computationally more complex)

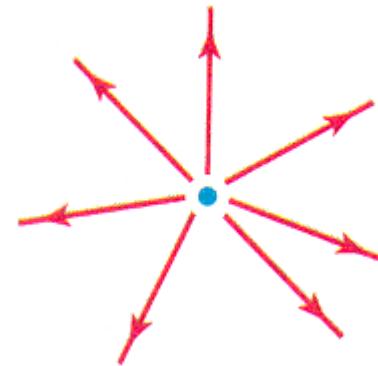
LIGHT SOURCES

Light Sources

- Objects **radiating light** and contributing to the illumination of a scene's objects
- Can be defined by several **features**:
 - Location
 - Color of emitted light
 - Emission direction
 - Shape

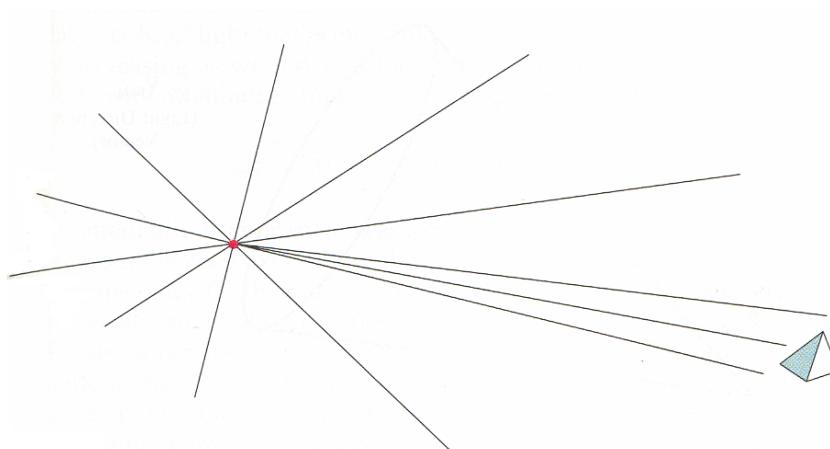


Simplified Light Sources



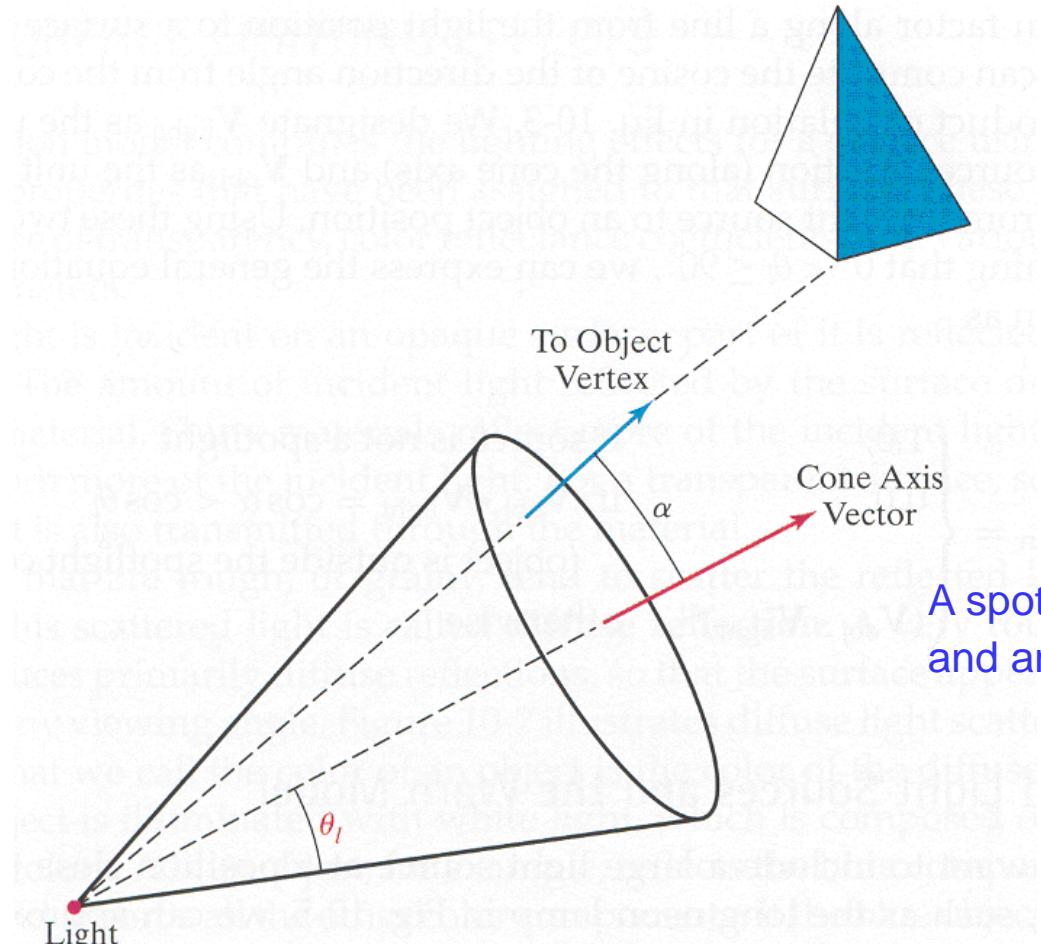
Isotropic point light source

Light source at an indefinite distance



Rays emitted by a light source at a far-away location can be considered as **parallel**

Spotlight



A spotlight is defined by a **direction** and an **emission angle**

SURFACE FEATURES

Surface Features

- An illumination model takes into account a **surface's optical properties**:

- reflection coefficients for each color
 - degree of transparency
 - texture parameters



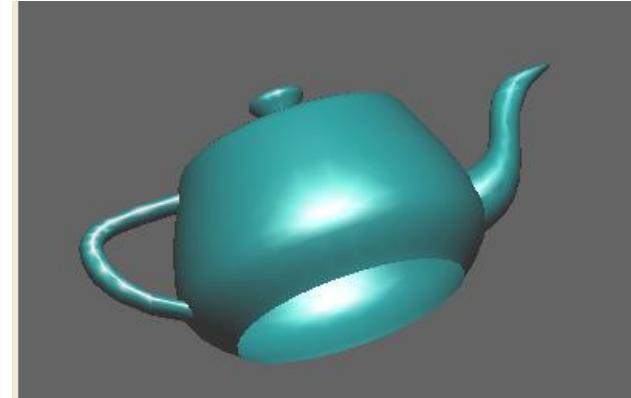
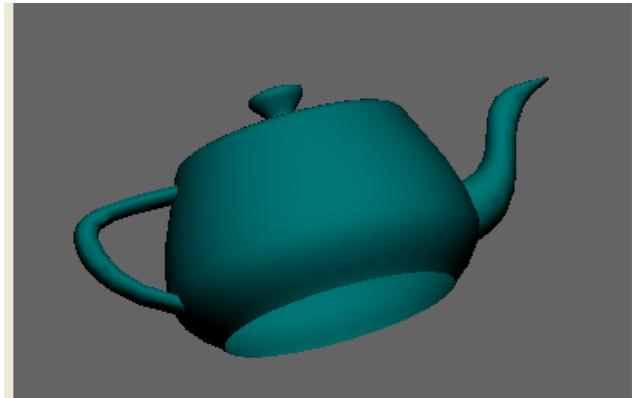
- When light is incident on an **opaque surface**:
 - part is **absorbed**
 - part is **reflected**

Surface Features



Surface Features

- The amount of reflected light depends on the **surface's features**
 - **Shiny surfaces** reflect more light
 - **Mate / dull surfaces** reflect less light

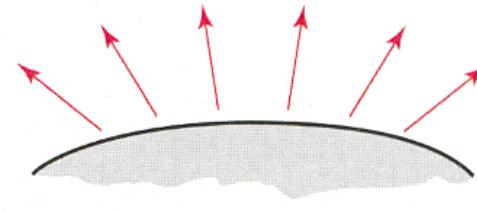


- **Transparent surfaces** transmit some light

Rough vs Smooth Surfaces

- Rough surfaces tend to spread the reflected light in all directions

- diffuse reflection

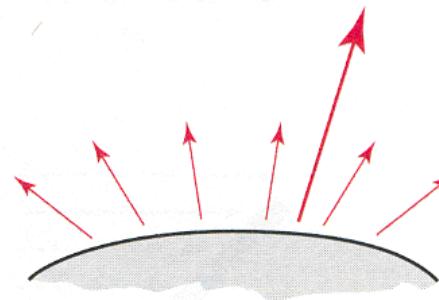


And look equally shiny from any viewpoint

- Smooth surfaces reflect more light in particular directions

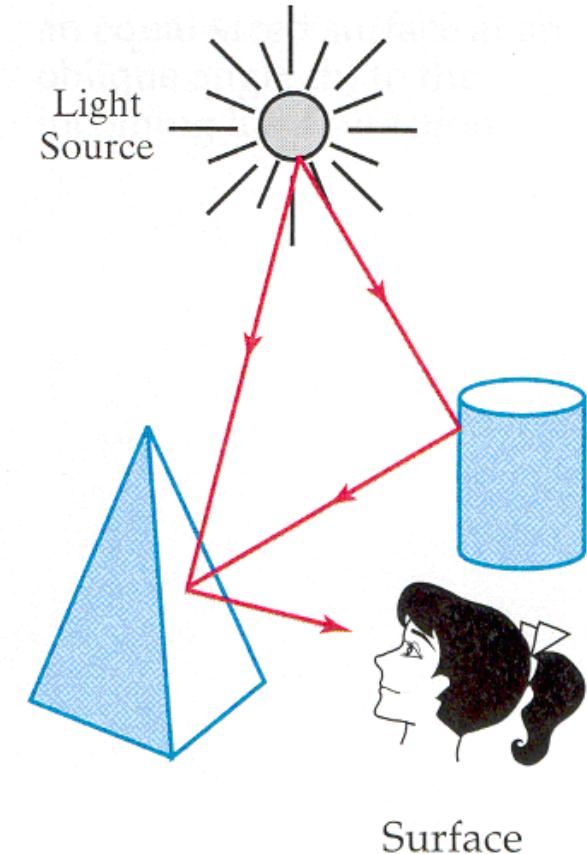
- specular reflection (*highlight*)

And present some shinier areas



Inter-Reflections

- Inter-reflections among scene objects might be approximated by
 - an **ambient illumination** component
- A surface **might not be directly illuminated** and still **be visible**, due to light reflected by other objects in the scene
- The amount of light reflected by a surface is the **sum of all contributions** from the light sources and the ambient illumination

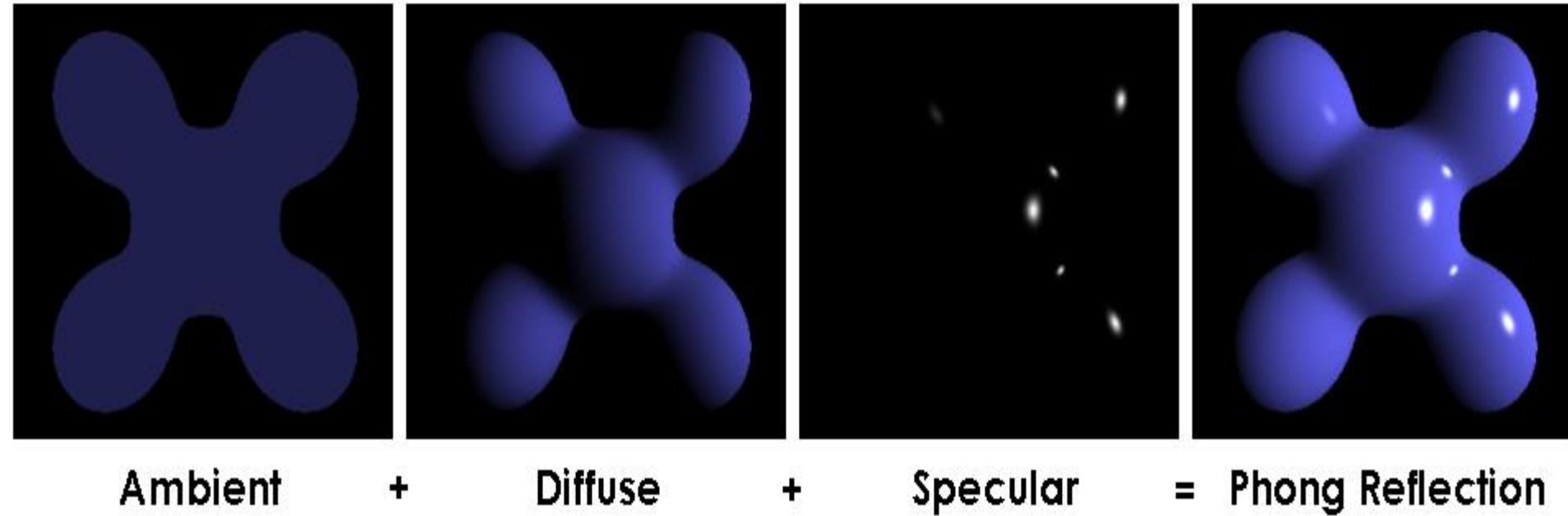


PHONG'S REFLECTION MODEL

Basic Illumination Models

- Sophisticated illumination models precisely compute the interaction effects between the radiating energy and the surface material
- Basic models use **approximations** to represent the physical processes producing the illumination effects
- The **empirical model** described next computes good enough results for most situations and includes:
 - ambient illumination
 - diffuse reflection
 - specular reflection

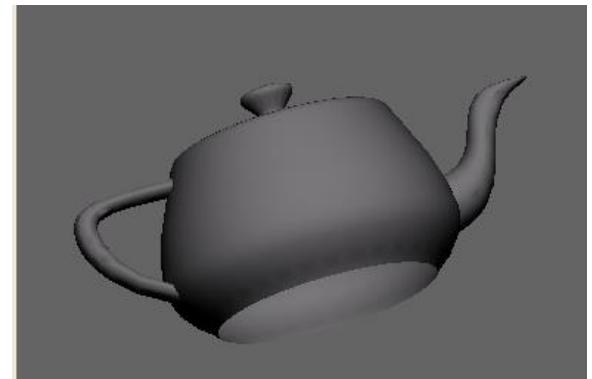
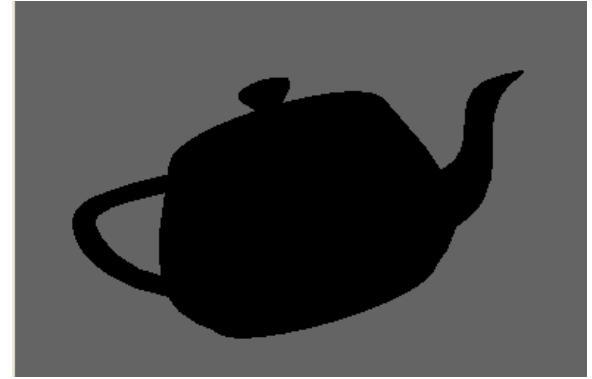
Phong reflection model – 1973



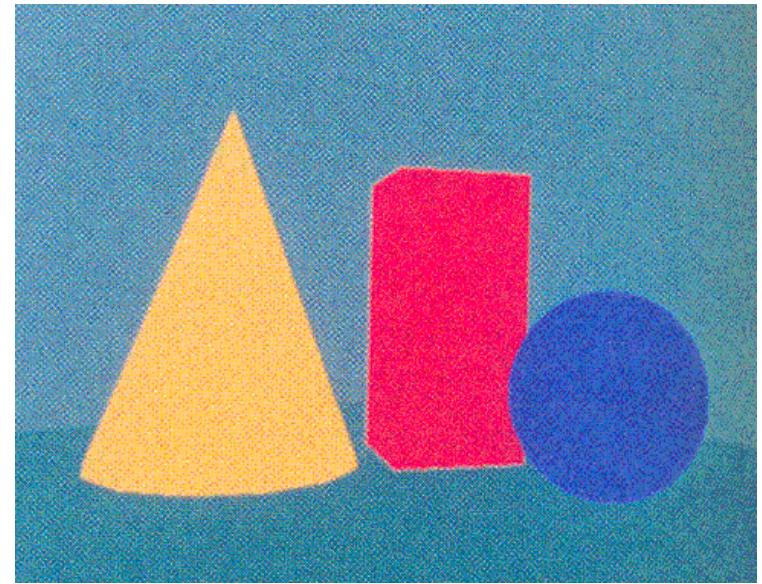
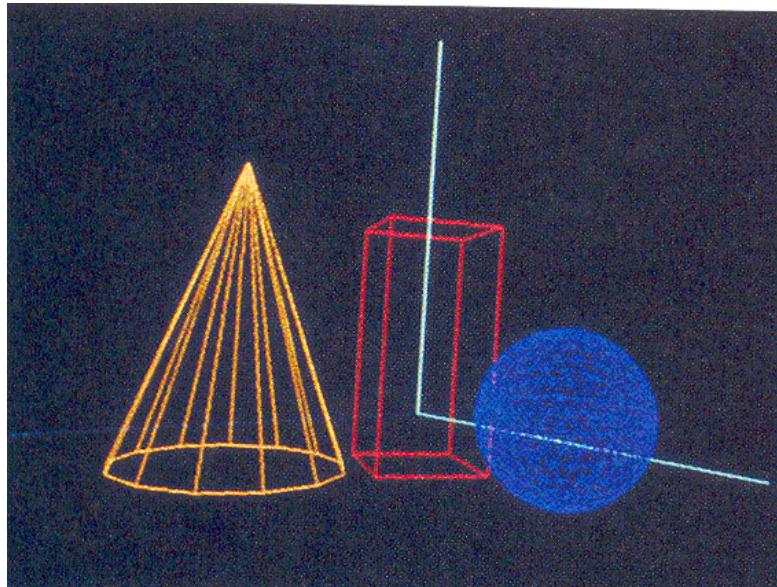
[Wikipedia]

Phong Model – Ambient illumination

- Constant illumination component for each model
- Independent from viewer position or object orientation !
- Take only material properties into account !

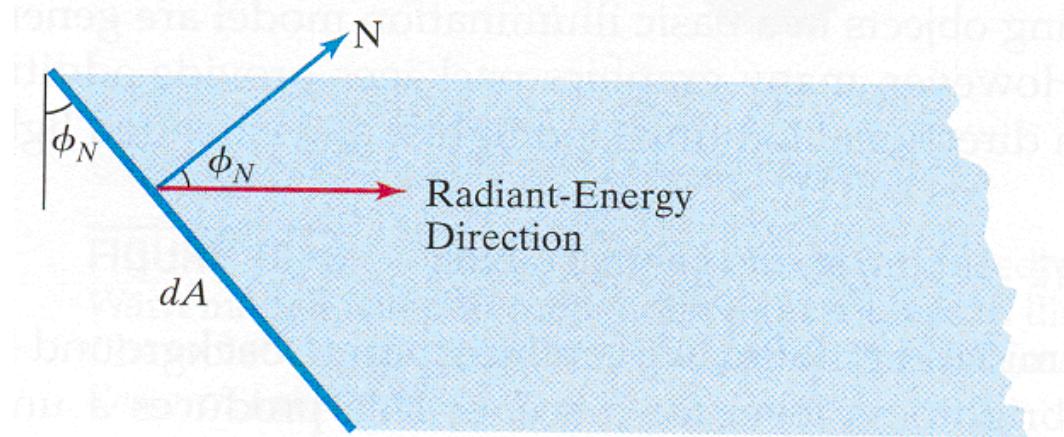


Phong Model – Ambient illumination



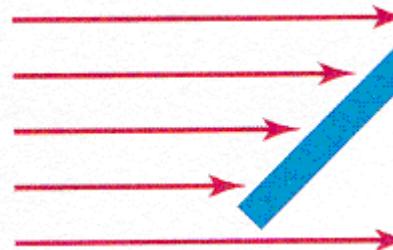
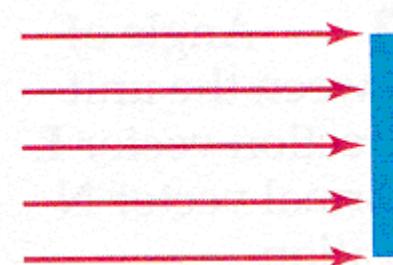
Diffuse Reflection

- It is considered that **incident light is spread with equal intensity in all directions**, regardless of the viewing direction
- Surfaces with that feature are called **Lambertian reflectors or ideal diffuse reflectors**
- The intensity of reflected light is computed by **Lambert's Law**:



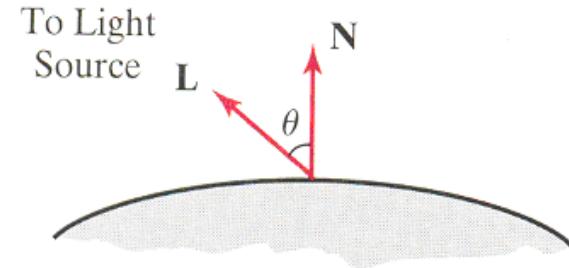
Diffuse Reflection

- There is, at least, **one point light source** (usually located at the viewpoint)
- The amount of incident light depends on the **surface orientation** regarding the **direction of the light source**
- A surface that is **orthogonal** to the light direction is “more illuminated” than an **oblique** surface, with the **same area**



Phong Model – Diffuse reflection

$$I_{l,\text{diff}} = \begin{cases} k_d I_l (\mathbf{N} \cdot \mathbf{L}), & \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$

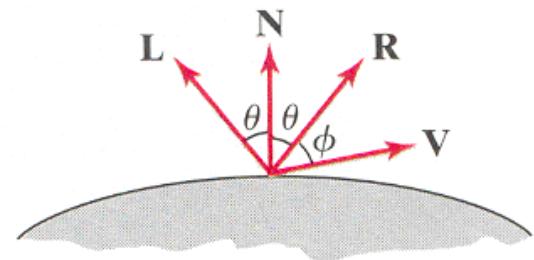


- Model surface is an **ideal diffuse reflector**
 - What does that mean ?
- No dependency from viewer position !
- Unit vectors !!

Phong Model – Specular reflection

- The shinier areas, **specular reflections** or *highlights*, that can be seen on shiny surfaces, result from the reflection of most light around concentrated areas

- The **specular reflection angle** is equal to the incidence angle (relative to the normal)
- **R** is the **unit vector** defining the ideal specular reflection direction
- **V** is the **unit vector** defining the viewing direction
- An **ideal reflector** reflects light only in the specular reflection direction (the viewer perceives the specular reflection only if **V** and **R** are coincident $\Phi = 0$)

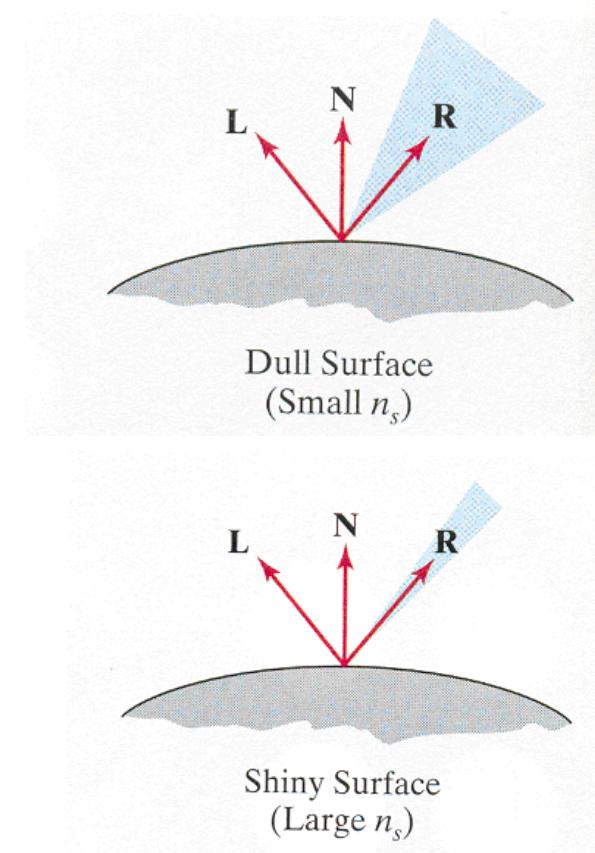


Phong Model – Specular reflection

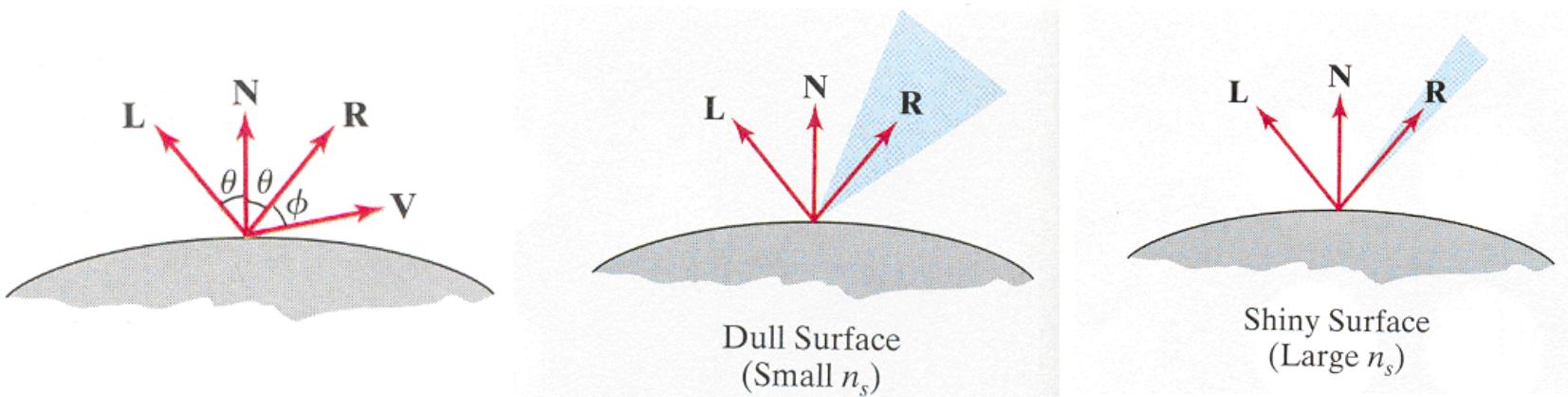
- Objects that are not ideal reflectors produce specular reflections in a **finite set of directions** around vector \mathbf{R}
- Shiny** surfaces have a **narrow** set of reflection directions
- The empirical **Phong specular reflection model** sets the intensity of the specular reflections as proportional to $\cos^{n_s} \phi$

$$I_{l,\text{spec}} = W(\theta) I_l \cos^{n_s} \phi$$

$W(\theta)$ is the **specular reflection coefficient**

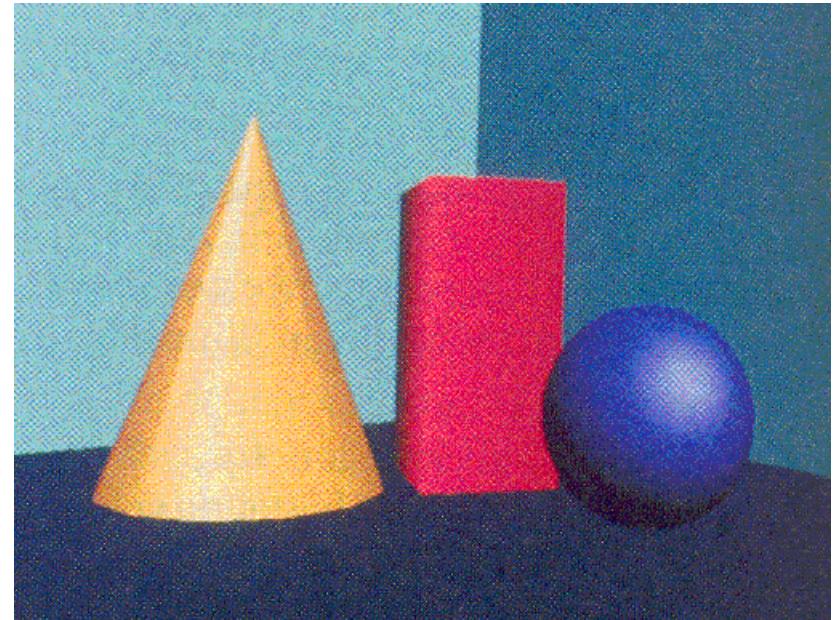
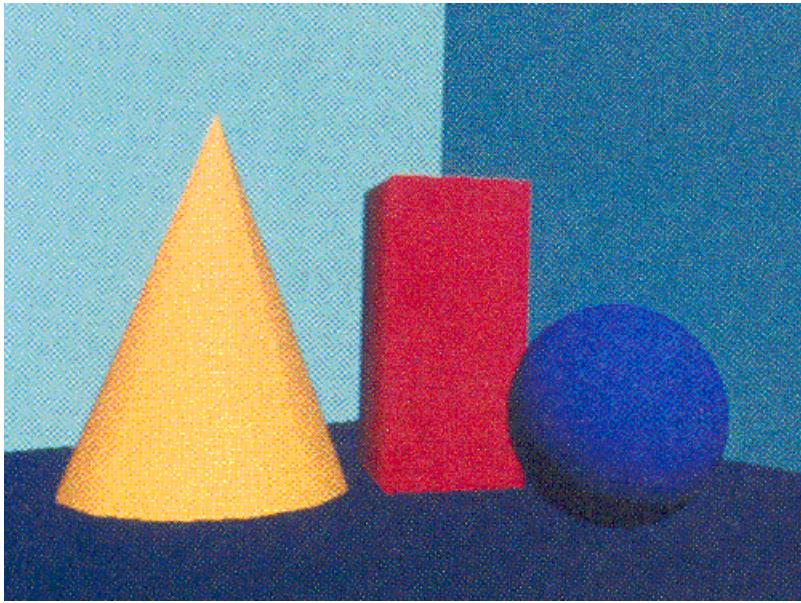


Phong Model – Specular reflection

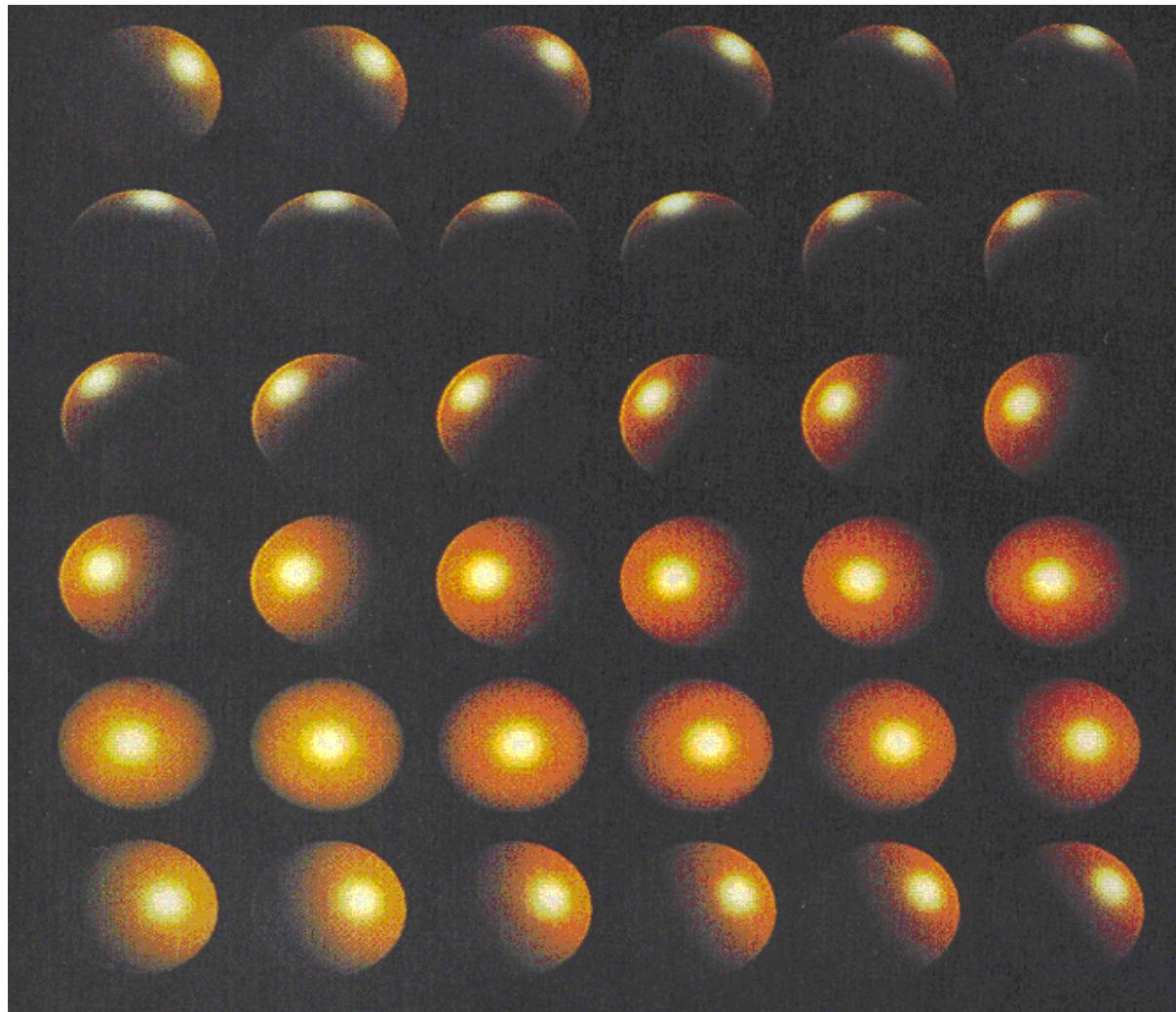


- Important for shiny model surfaces
 - How to model **shininess** ?
- Take into account **viewer position** !
- Unit vectors !!

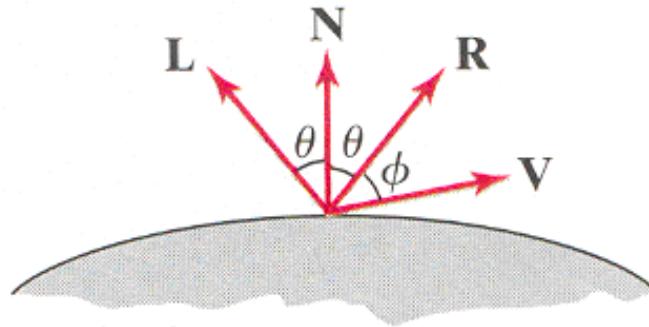
Phong Model – Specular reflection



Phong Model – Specular reflection

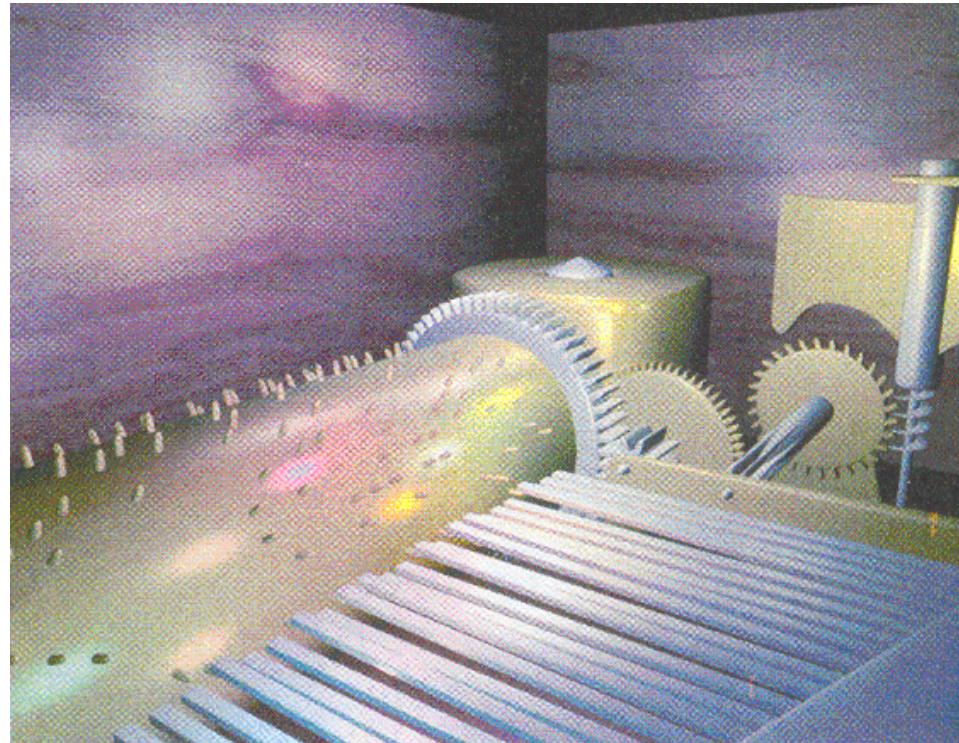


Phong Model – Specular reflection



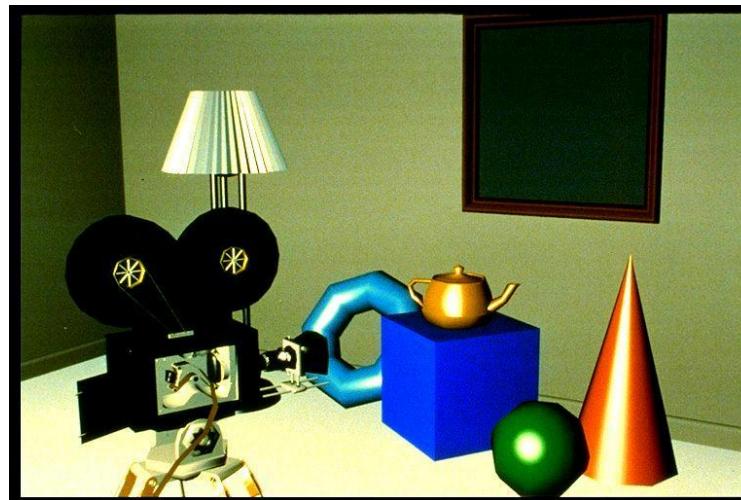
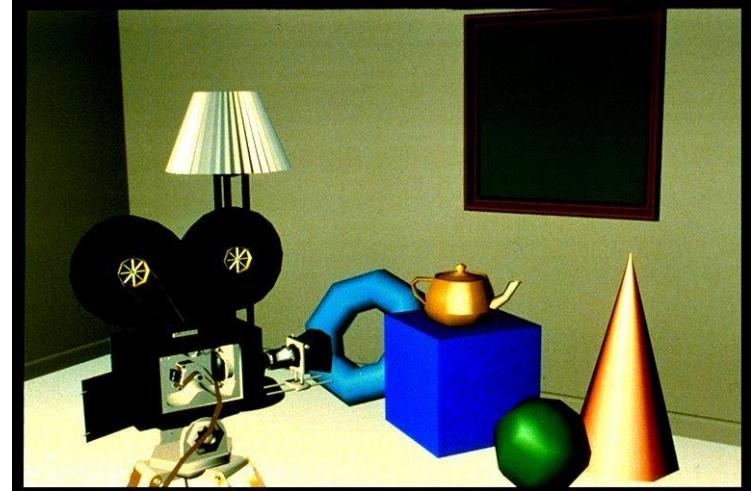
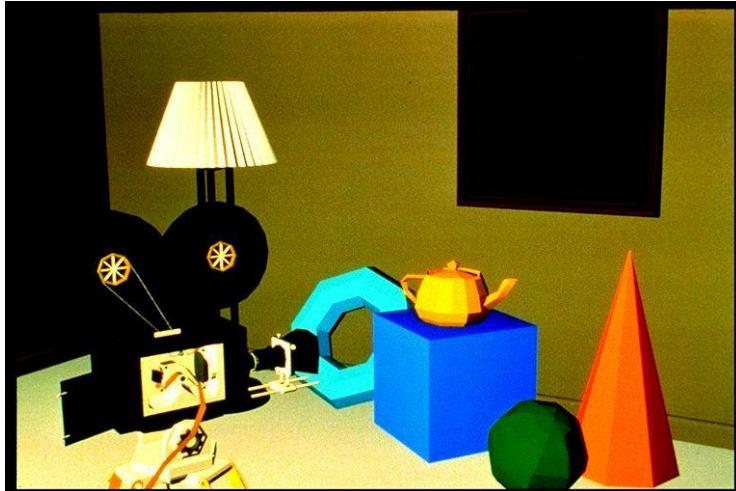
$$I_{l,\text{spec}} = \begin{cases} k_s I_l (\mathbf{V} \cdot \mathbf{R})^{n_s}, & \text{if } \mathbf{V} \cdot \mathbf{R} > 0 \quad \text{and} \quad \mathbf{N} \cdot \mathbf{L} > 0 \\ 0.0, & \text{if } \mathbf{V} \cdot \mathbf{R} < 0 \quad \text{or} \quad \mathbf{N} \cdot \mathbf{L} \leq 0 \end{cases}$$

More than one light source



$$I = k_a I_a + \sum_{l=1}^n I_l [k_d (\mathbf{N} \cdot \mathbf{L}) + k_s (\mathbf{N} \cdot \mathbf{H})^{n_s}]$$

Can you spot the differences ?



Material properties



[OpenGL – The Red Book]

Material properties

Material	ambient (k_a)	diffuse (k_d)	specular (k_s)	specular exponent (m)	translucency (a)
Brass	0.329412 0.223529 0.027451	0.780392 0.568627 0.113725	0.992157 0.941176 0.807843	27.8974	1.0
Bronze	0.2125 0.1275 0.054	0.714 0.4284 0.18144	0.393548 0.271906 0.166721	25.6	1.0
Polished Bronze	0.25 0.148 0.06475	0.4 0.2368 0.1036	0.774597 0.458561 0.200621	76.8	1.0
Chrome	0.25 0.25 0.25	0.4 0.4 0.4	0.774597 0.774597 0.774597	76.8	1.0
Copper	0.19125 0.0735 0.0225	0.7038 0.27048 0.0828	0.256777 0.137622 0.086014	12.8	1.0
Polished Copper	0.2295 0.08825 0.0275	0.5508 0.2118 0.066	0.580594 0.223257 0.0695701	51.2	1.0
Gold	0.24725 0.1995 0.0745	0.75164 0.60648 0.22648	0.628281 0.555802 0.366065	51.2	1.0
Polished Gold	0.24725 0.2245 0.0645	0.34615 0.3143 0.0903	0.797357 0.723991 0.208006	83.2	1.0

[<https://people.eecs.ku.edu/~jrmiller/Courses/672/InClass/3DLighting/MaterialProperties.html>]

MORE SOPHISTICATED REFLECTION MODELS

Cook & Torrance, 1982



Other Illumination Models

- Models you have seen before are **empirical**, but look okay
 - Phong Model
 - Blinn-Phong Model
- More complex models are **physically based**
 - Cook-Torrance Model
 - Oren-Nayer Model
- **Performance vs. accuracy** tradeoff

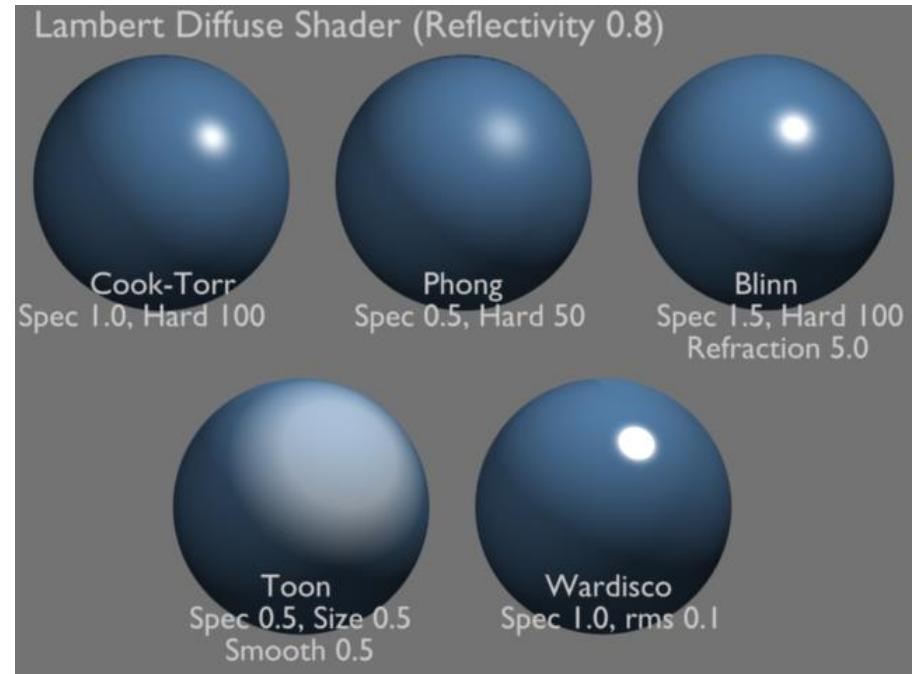


Image credit:

<http://wiki.blender.org/index.php/File:Manual-Shaders-Lambert.png>

[Andy Van Dam]

SHADING

Illumination and Shading

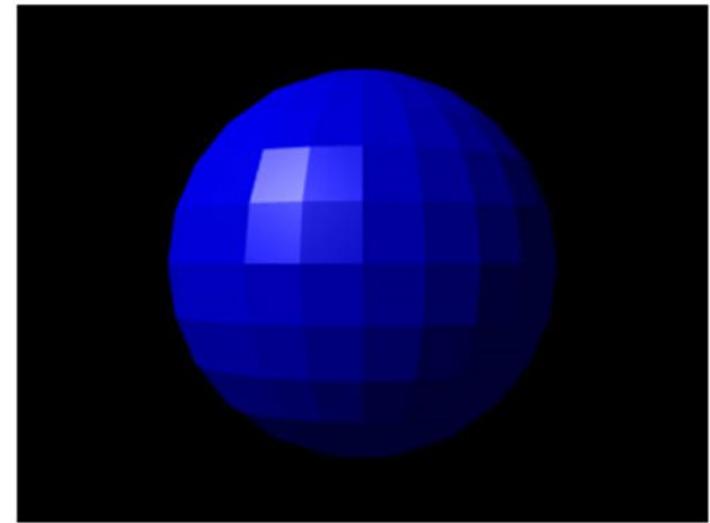
- Color values resulting from an illumination model can be used for **surface rendering** in different ways:
 - Compute the color values **for each and every pixel** corresponding to the projected surface
 - Compute the color values just **for a few chosen pixels**, and compute **approximate color values** for the rest
- In general, graphics APIs use ***scan-line* algorithms** and use the illumination model to compute **color values at mesh vertices**
 - Some **interpolate color values along the *scan-lines***
 - Other use more precise methods

Illumination and Shading

- How to optimize?
 - Fewer light sources
 - Simple **shading** method
- BUT, less computations mean less realism
 - Wireframe representation
 - Flat-**shading**
 - Gouraud **shading**
 - Phong **shading**

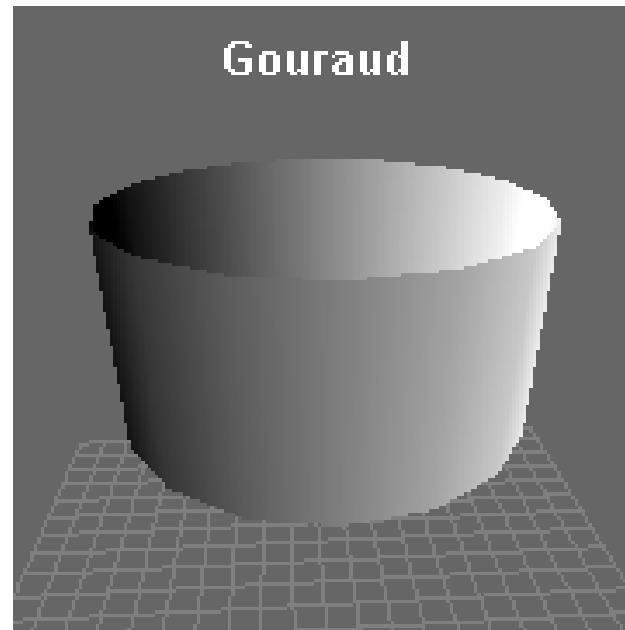
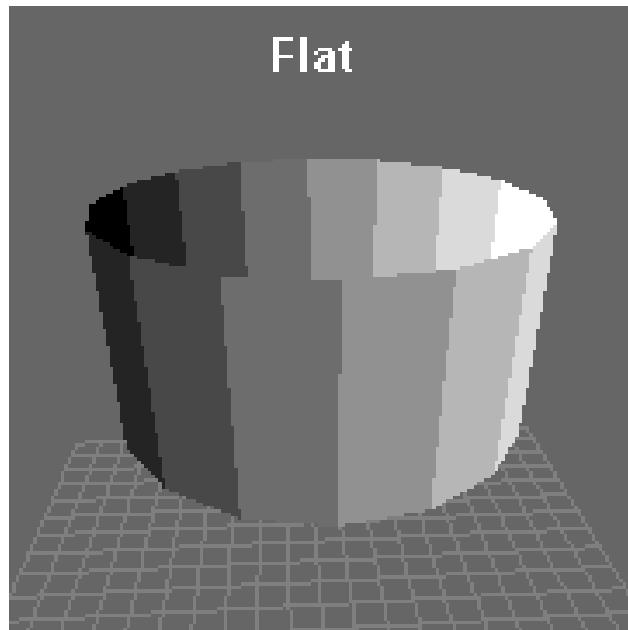
Flat-Shading

- For each triangle / polygon
 - Apply the illumination model **just once !**
 - All **pixels** have the **same color**
- Fast !
- But objects seem “**blocky**”



FLAT SHADING

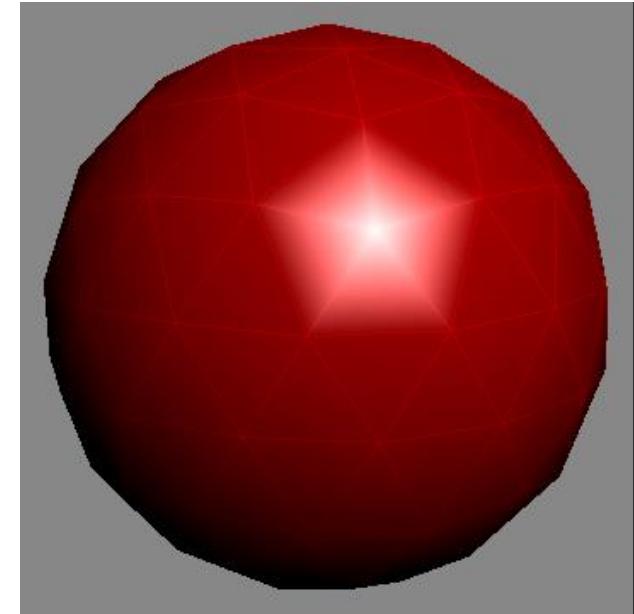
Flat-Shading vs Gouraud Shading



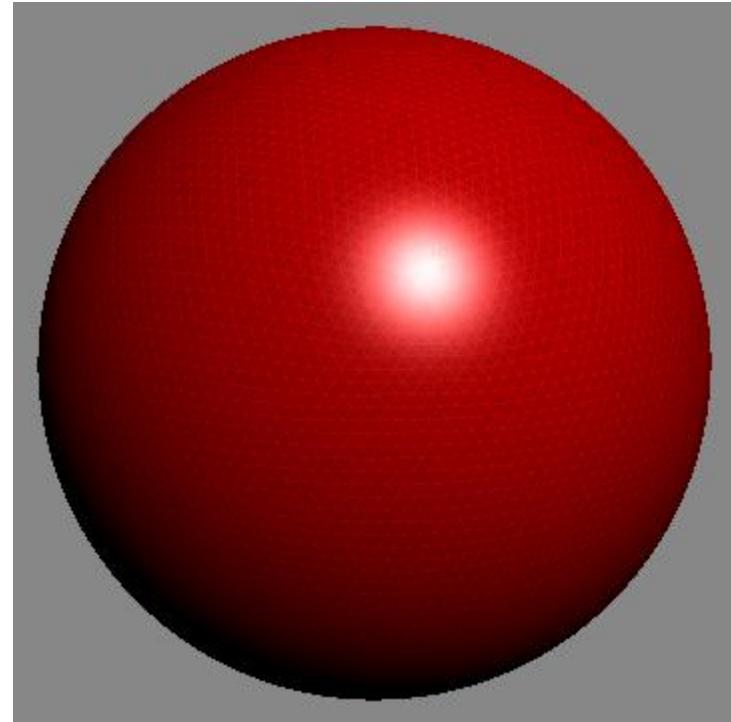
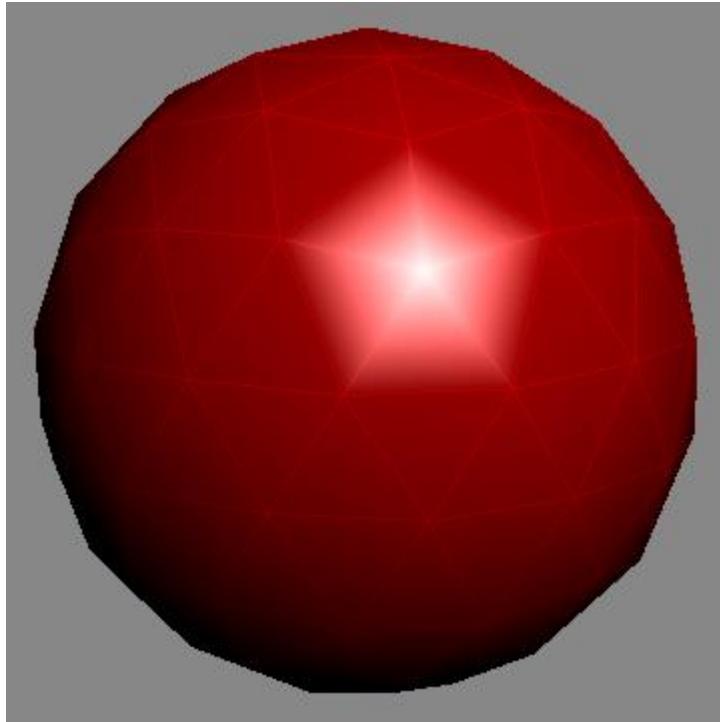
[Wikipedia]

Gouraud Shading – 1971

- For each triangle / polygon
 - Apply the illumination model at **each vertex**
 - **Interpolate** color to shade each pixel
- Better than flat-shading
- Problems with highlights
- Mach-effect



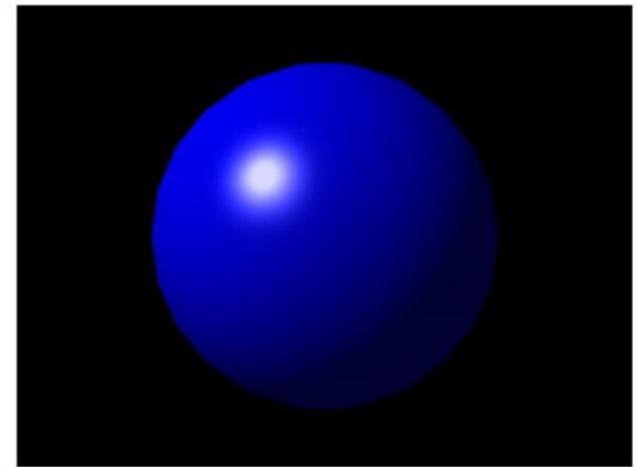
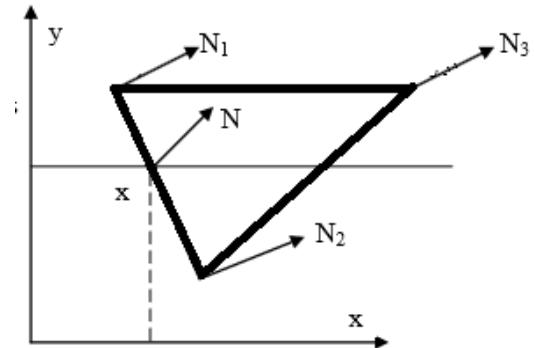
Gouraud Shading – More triangles !



[Wikipedia]

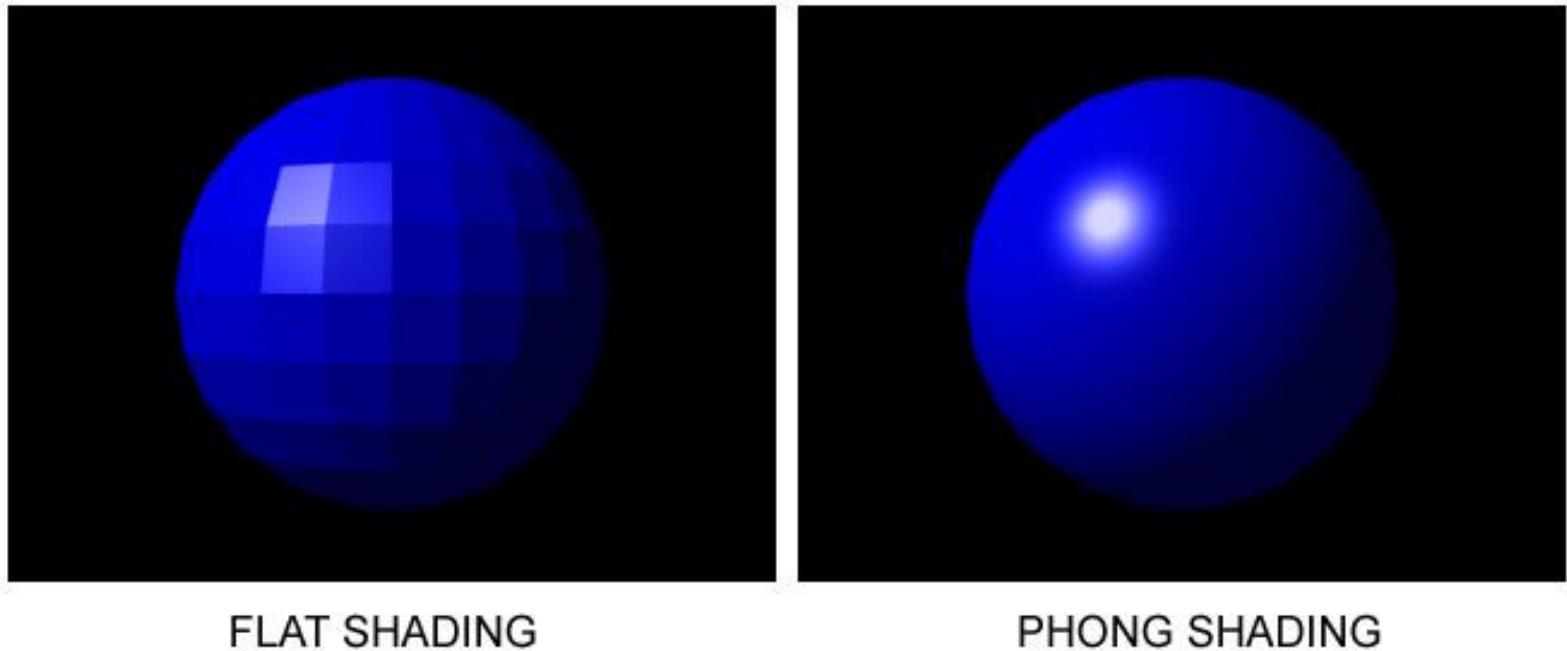
Phong Shading – 1973

- For each triangle / polygon
 - Interpolate normal vectors across rasterized polygons
- Better than Gouraud shading
- BUT, more time consuming



PHONG SHADING

Flat-Shading vs Phong Shading



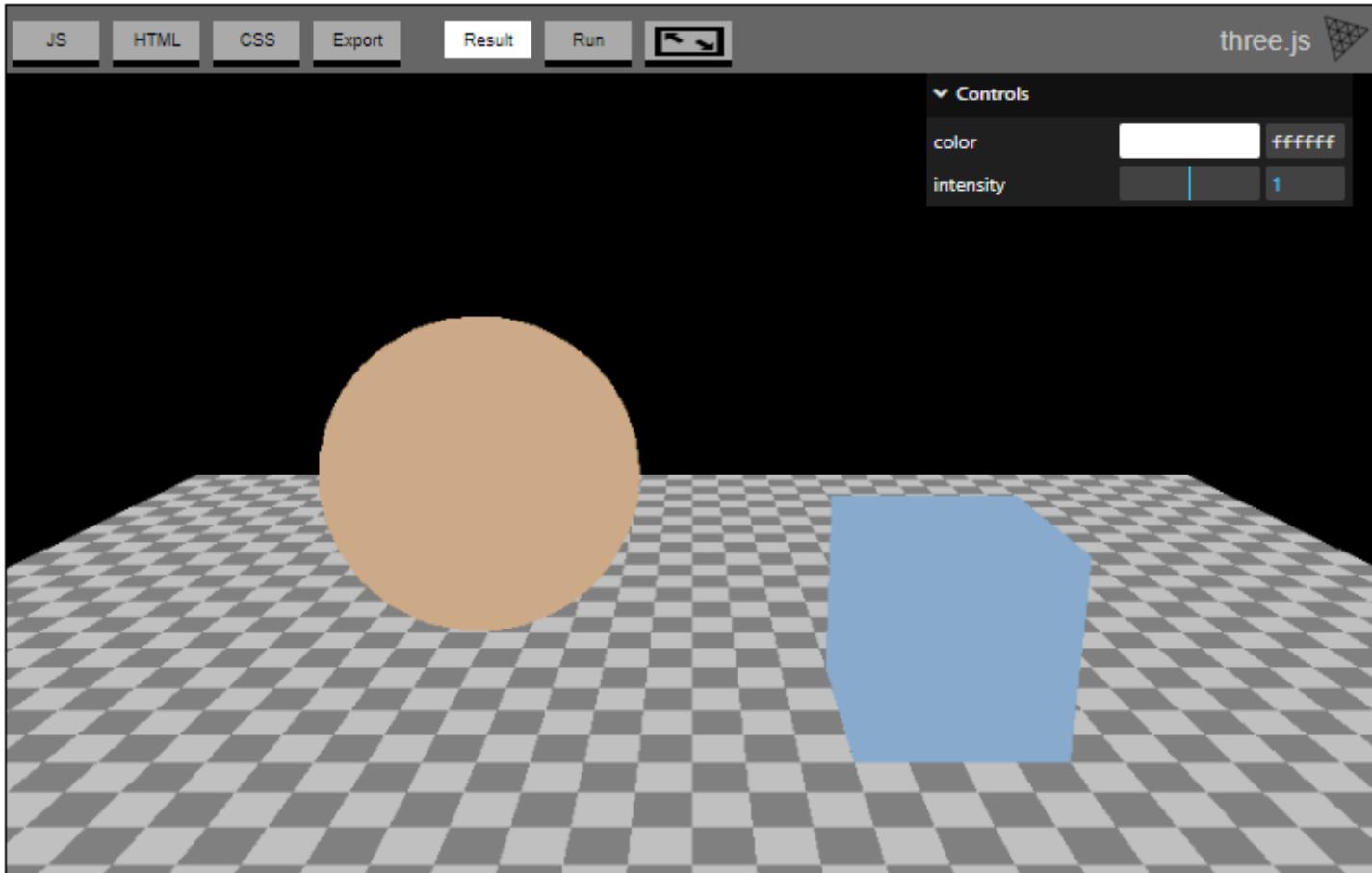
[Wikipedia]

Possible References

- The basics of illumination and shading are presented in any Computer Graphics book
- E. Angel and D. Shreiner. Interactive Computer Graphics, 7th Ed., Addison-Wesley, 2015
- J M Pereira, et al. Introdução à Computação Gráfica. FCA, 2018

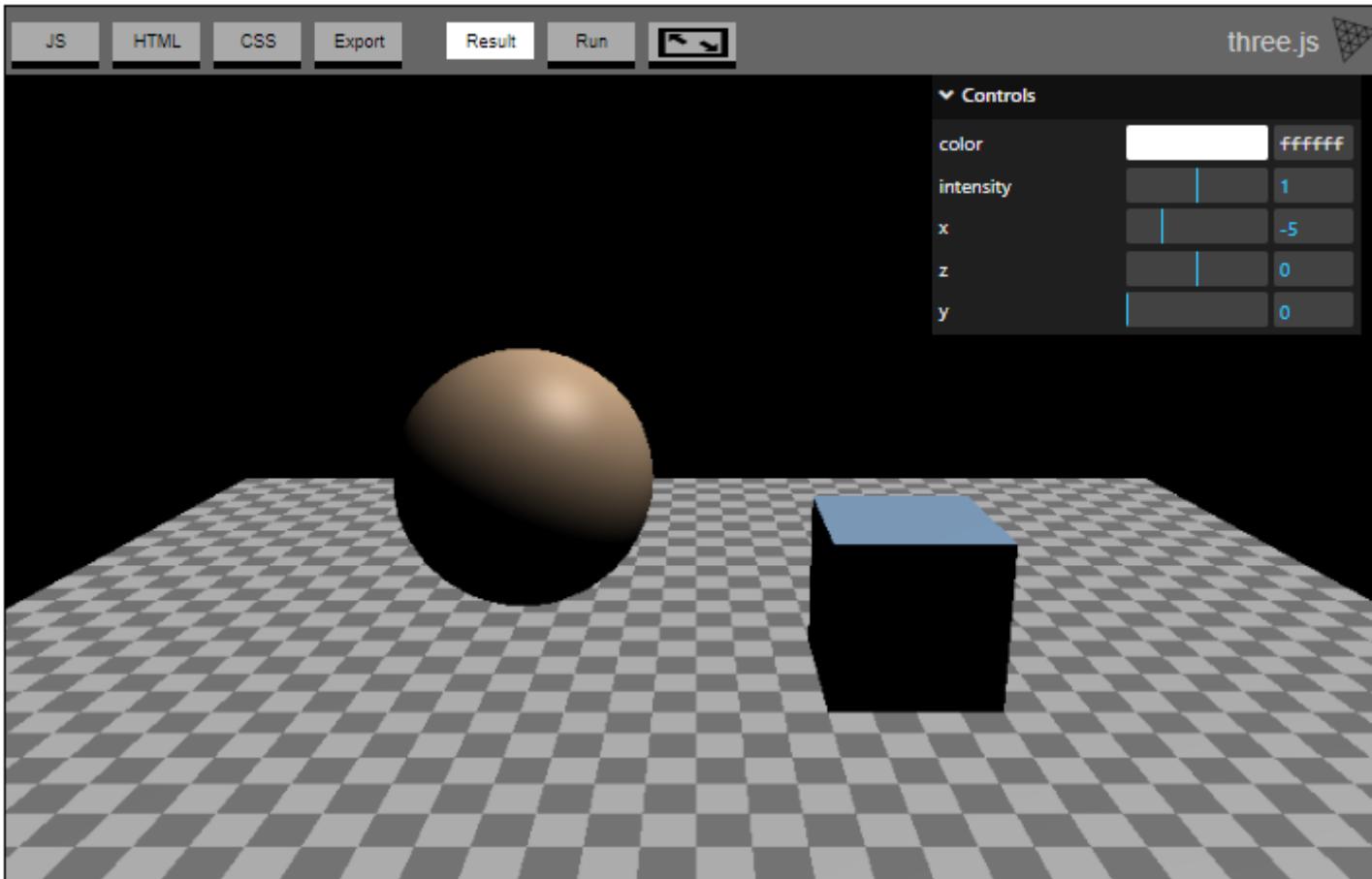
THREE.JS LIGHT SOURCES & SHADING

Three.js – AmbientLight



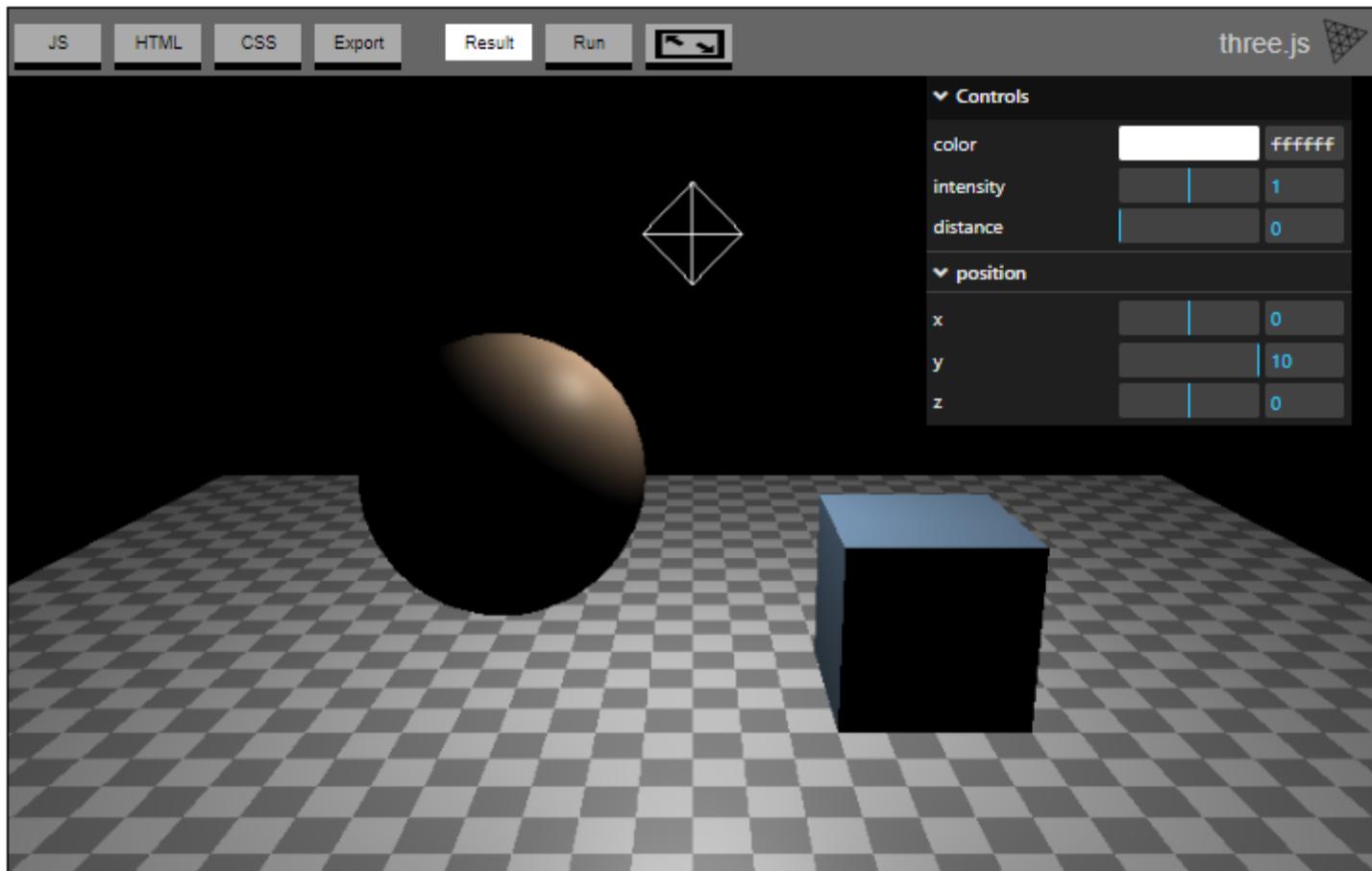
<https://threejs.org/manual/examples/lights-ambient.html>

Three.js – DirectionalLight



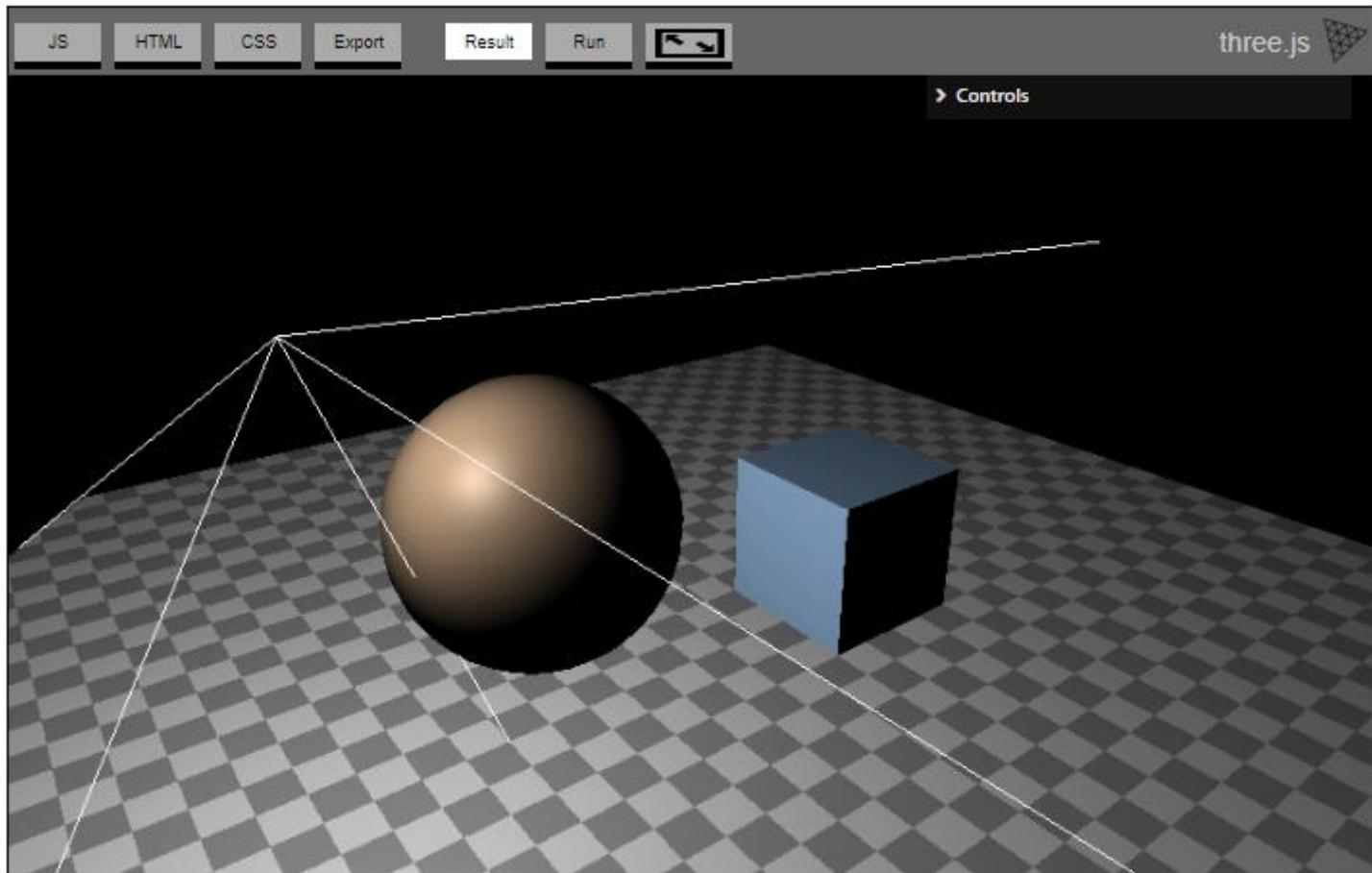
<https://threejs.org/manual/examples/lights-directional.html>

Three.js – PointLight



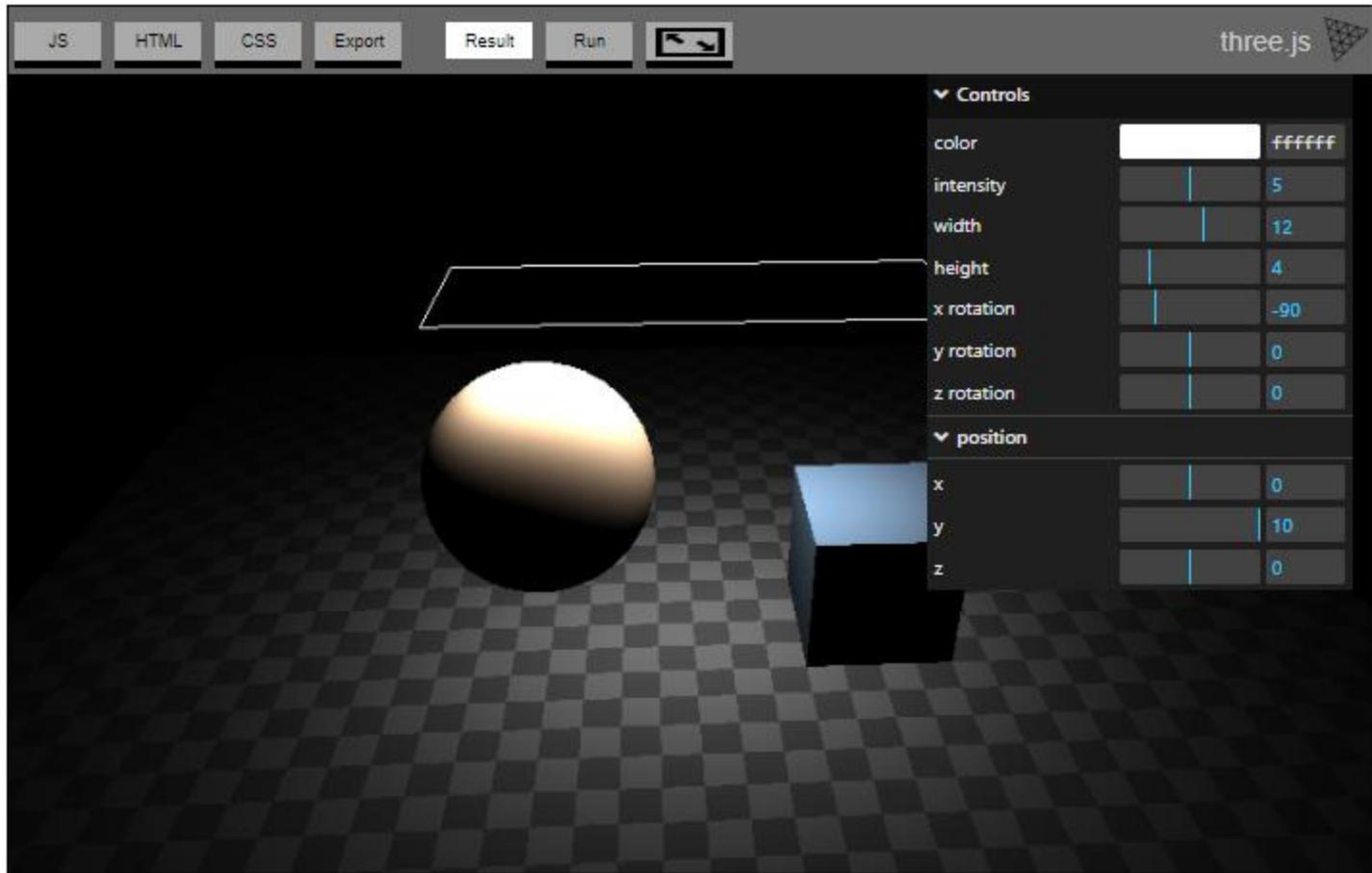
<https://threejs.org/manual/examples/lights-point.html>

Three.js – SpotLight



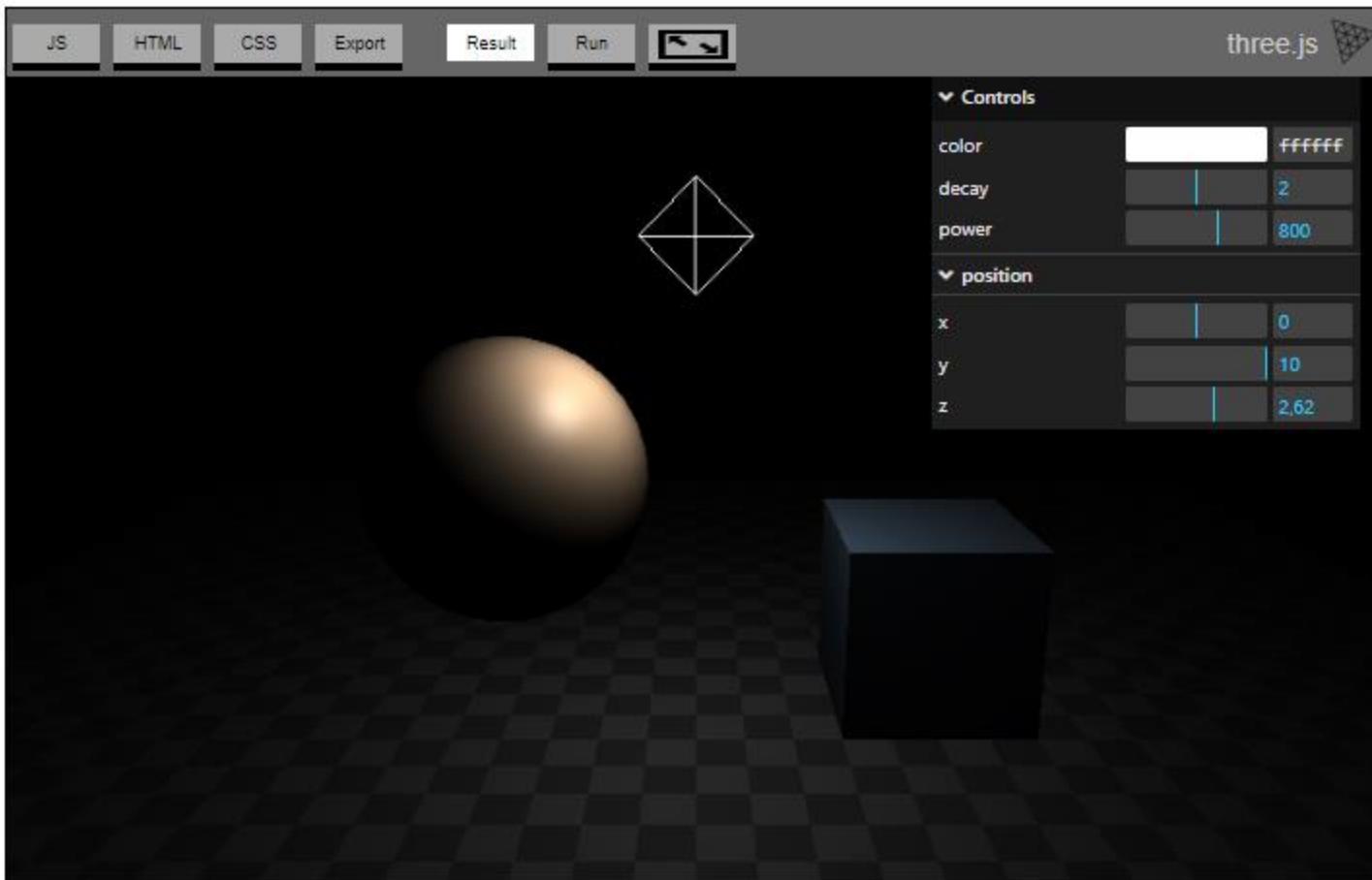
<https://threejs.org/manual/examples/lights-spot-w-helper.html>

Three.js – RectAreaLight



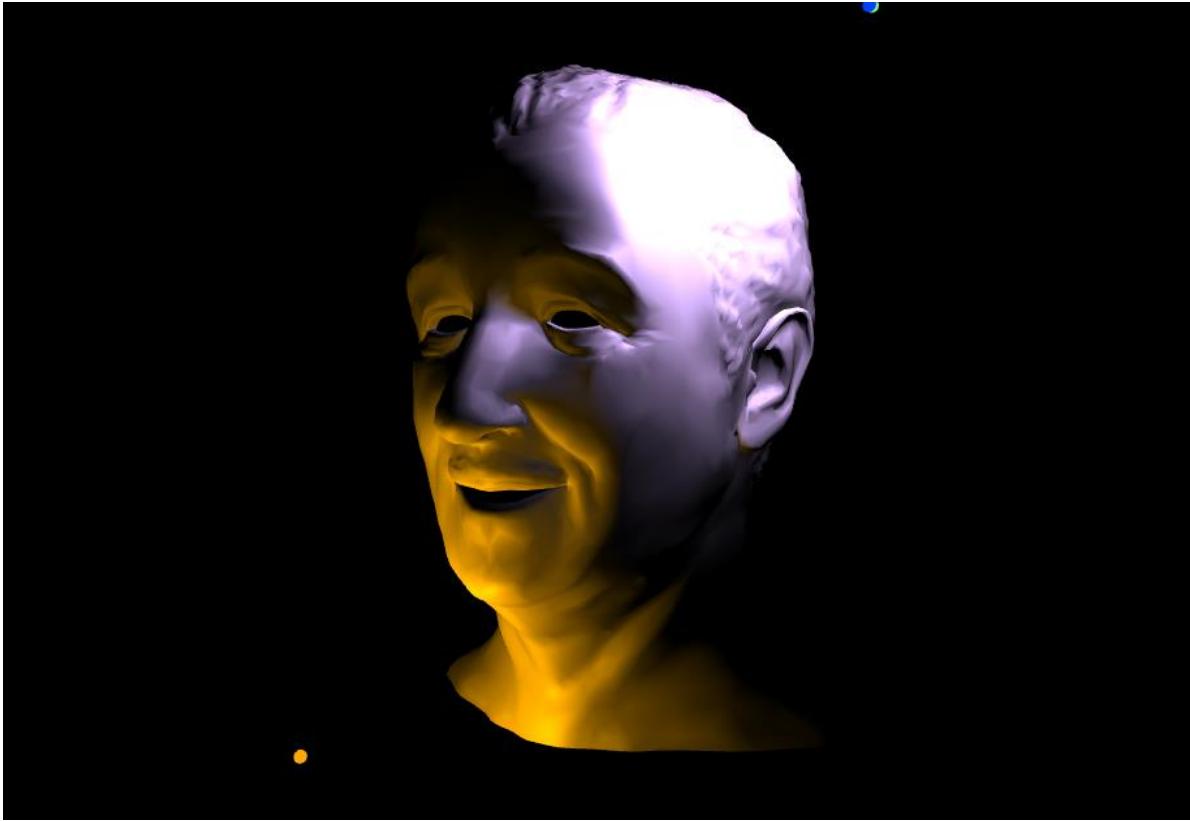
<https://threejs.org/manual/examples/lights-rectarea.html>

Three.js – Fade out with distance



<https://threejs.org/manual/examples/lights-point-physically-correct.html>

Three.js – Point light sources example

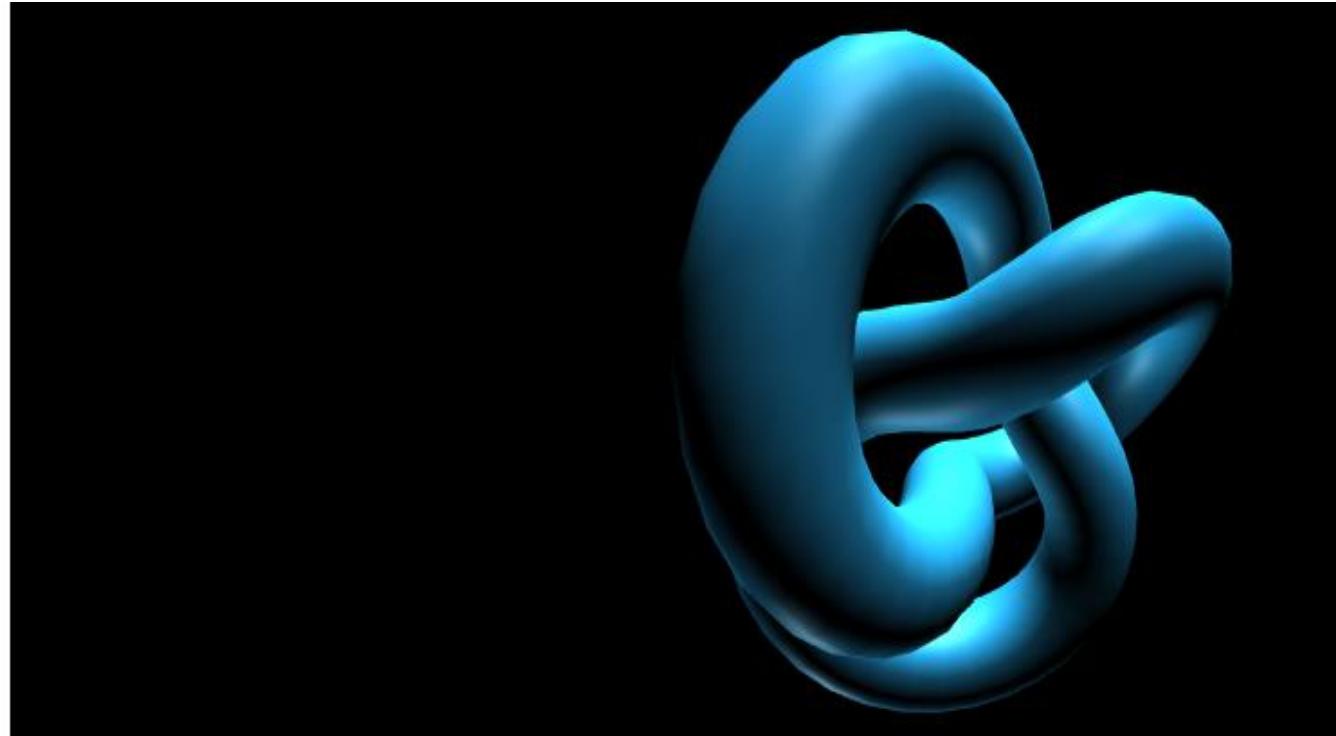


https://threejs.org/examples/#webgl_lights_pointlights

Three.js – Materials

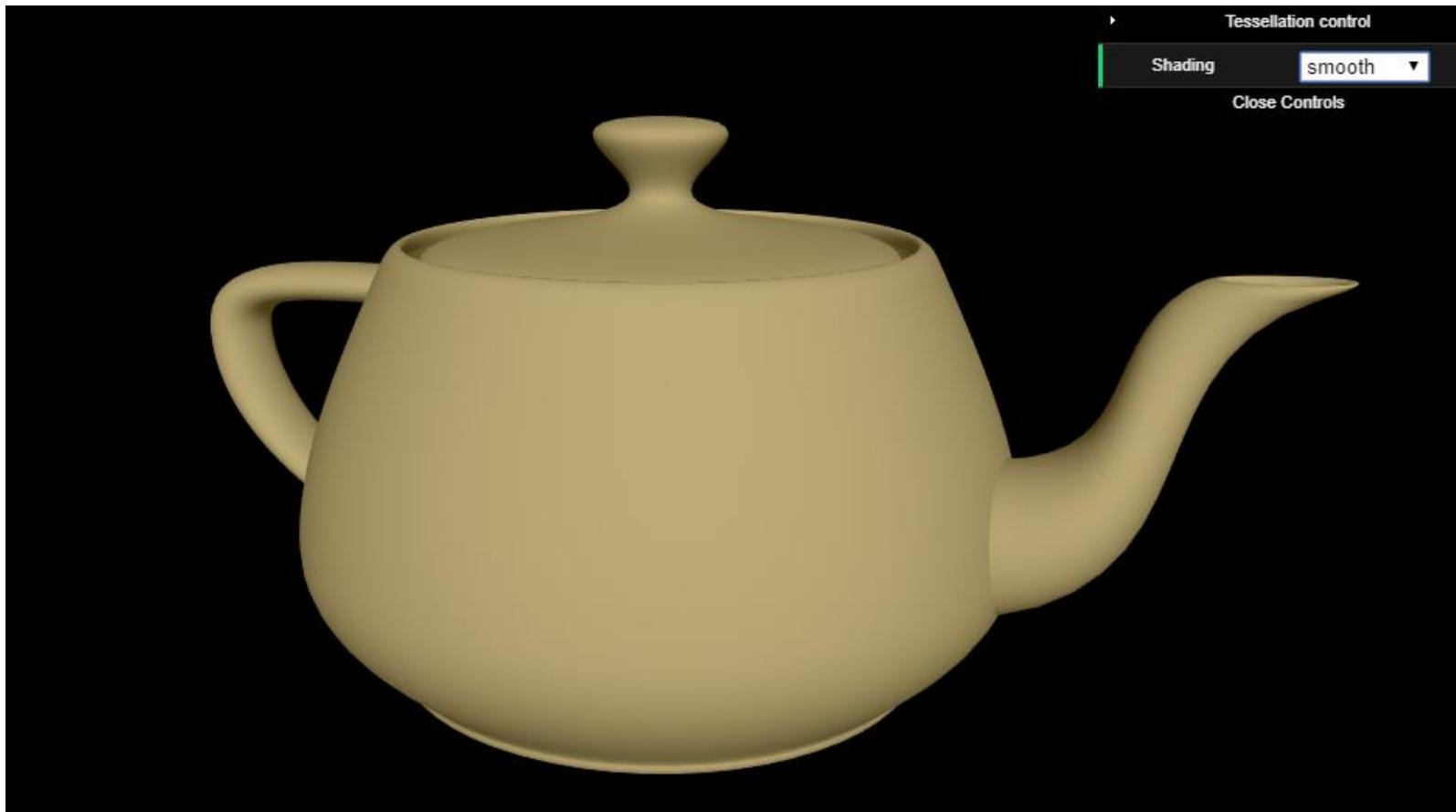
Materials

```
LineBasicMaterial  
LineDashedMaterial  
Material  
MeshBasicMaterial  
MeshDepthMaterial  
MeshLambertMaterial  
MeshNormalMaterial  
MeshPhongMaterial  
MeshPhysicalMaterial  
MeshStandardMaterial  
MeshToonMaterial  
PointsMaterial  
RawShaderMaterial  
ShaderMaterial  
ShadowMaterial  
SpriteMaterial
```



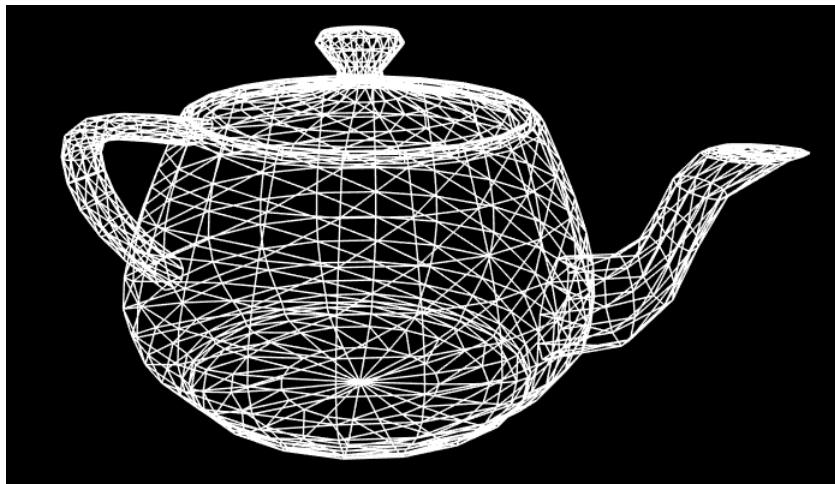
<https://threejs.org/docs/index.html#api/en/materials/MeshPhongMaterial>

Three.js – The Teapot



https://threejs.org/examples/#webgl_geometry_teapot

Three.js – Shading Examples



https://threejs.org/examples/#webgl_geometry_teapot

ACKNOWLEDGMENTS

Acknowledgments

- Some ideas and figures have been taken from slides of other CG courses.
- In particular, from the slides made available by Beatriz Sousa Santos, Ed Angel and Andy van Dam.
- Thanks!

An Introduction to Geometric Modeling using Polygonal Meshes

Joaquim Madeira

May 2022

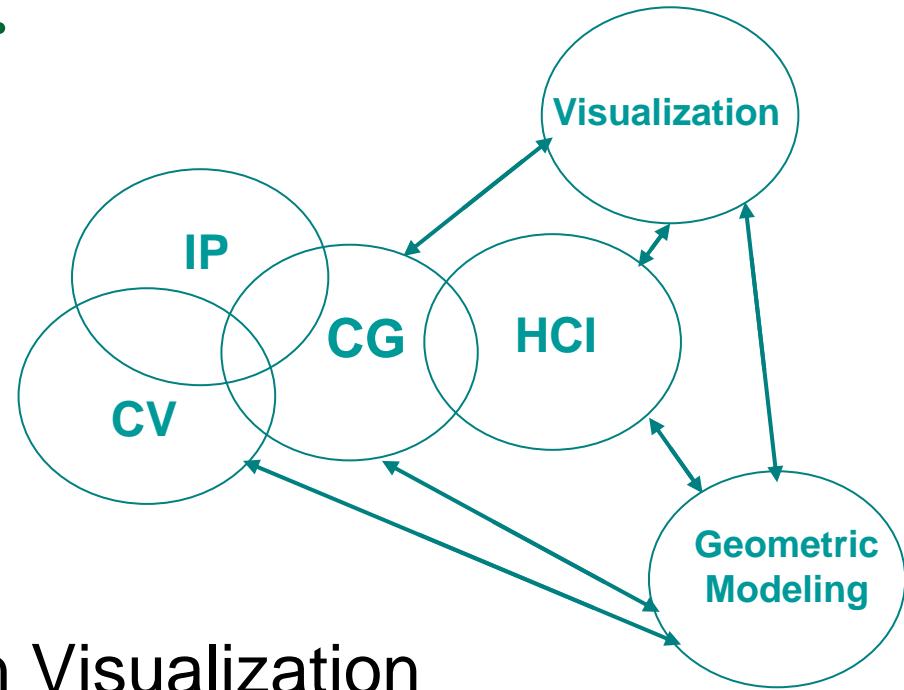
Overview

- Motivation
- Polygonal meshes
- Exact vs. Approximate representation
- Geometrical and topological information
- Valid vs. non-valid models
- The Euler formula
- Computational representation

COMPUTER GRAPHICS & GEOMETRIC MODELING

CG is not alone...

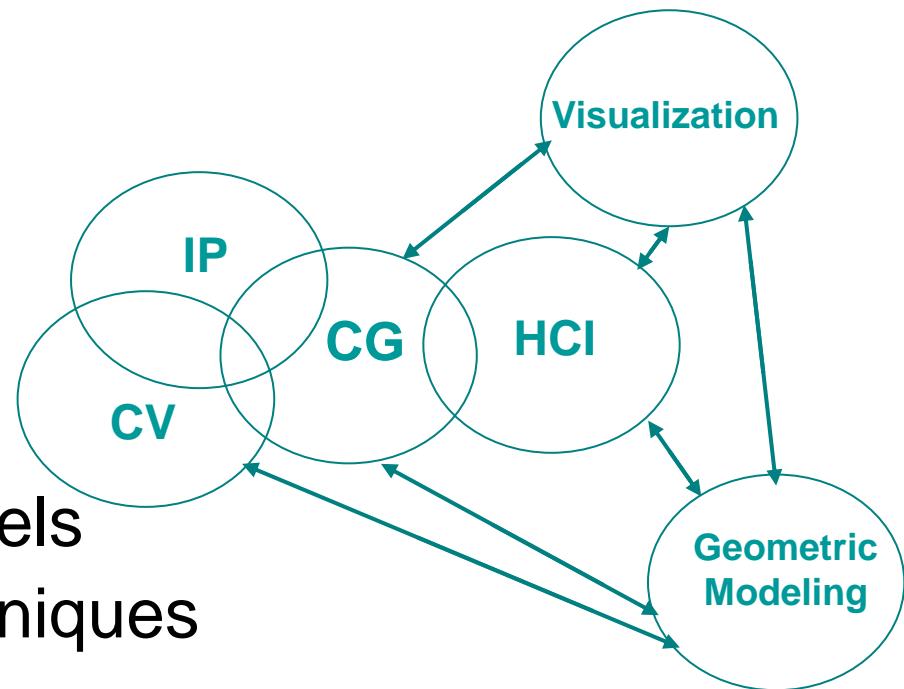
- Core areas:
 - CG, IP, CV and HCI
- Satellite areas:
 - Geometric Modeling
 - Data and Information Visualization
- What is common?
 - CG, IP : image file formats, color models, ...
 - CG, CV : 3D model representations, ...
 - IP, CV : noise removal, filters, ...



CG is not alone...

■ Geometric Modeling

- CV : 3D scanning
- CG : 2D and 3D models
- HCI : interaction techniques



■ Visualization

- HCI : interaction techniques
- GeoM : 2D and 3D models
- CG : rendering

Example – Medical Imaging

- Processing pipeline
 - Noise removal
 - Segmentation
 - Generating 2D / 3D models
 - Data visualization
 - User interaction
 - ...



[www.mevislab.de]

CG Main Tasks

■ Modeling

- Construct individual models / objects
- Assemble them into a 2D or 3D scene



■ Animation

- Static vs. dynamic scenes
- Movement and / or deformation

■ Rendering

- Generate final images
- Where is the observer?
- How is he / she looking at the scene?

Geometric Modeling

- A **geometric model** describes the **shape** of an (real or virtual) object
- How ?
 - Different mathematical representations ?
 - Data structures ?
 - Possible operations ?
 - Compactness ? Robustness ? Efficiency ?
 - Interpolation vs Approximation ?
 - ...

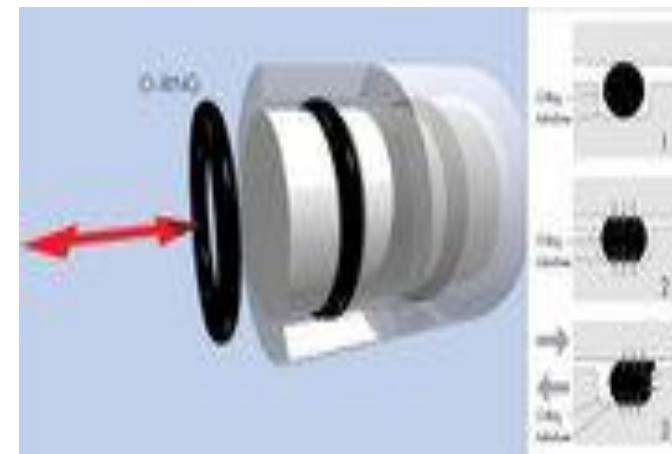
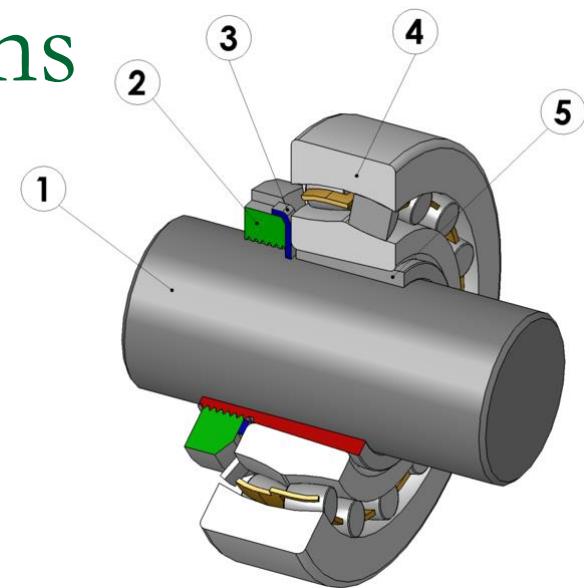
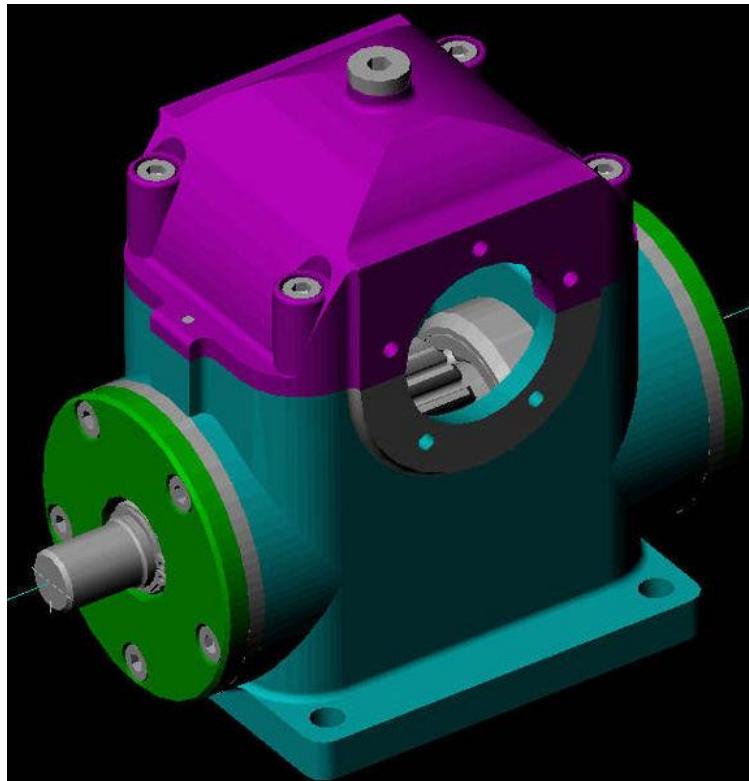
Geometric Modeling

■ What for ?

- Distinguish between **inside**, **outside** and **border** of a model
- Compute **properties**
 - Centroid
 - Area / Volume
 - ...
- Detect interferences / **collisions**
- Compute **light reflections** and / or transparencies
- ...

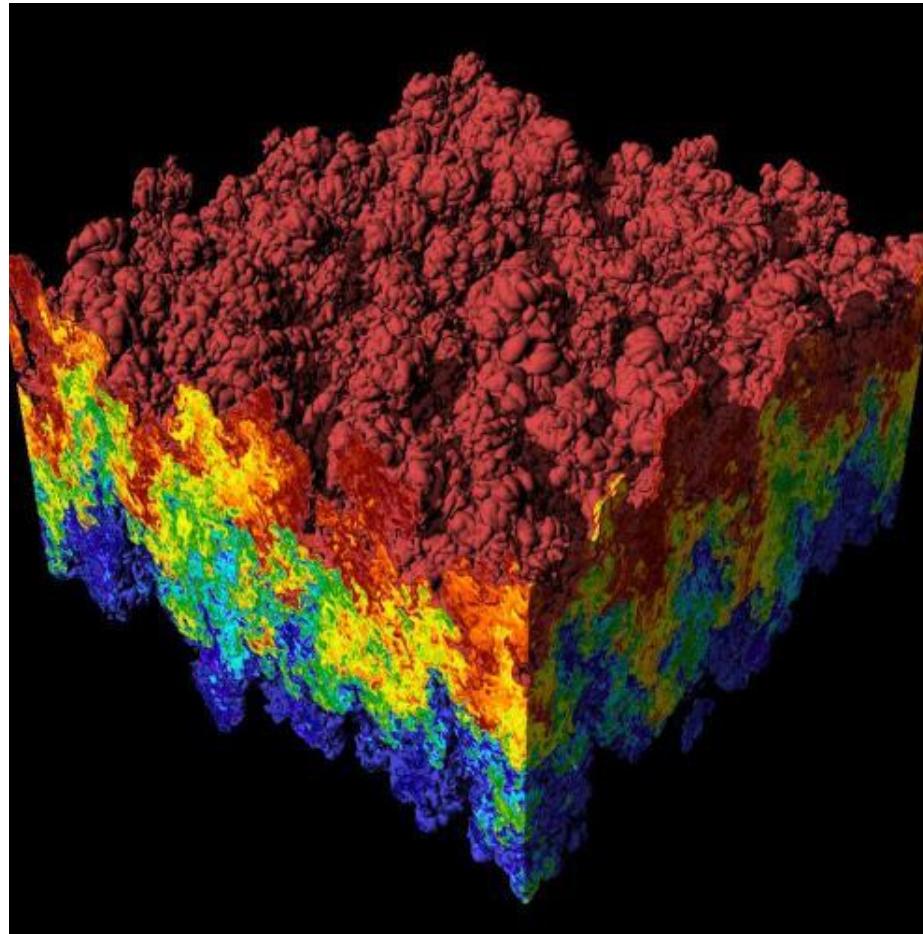
3D models – Applications

■ CAD / CAM



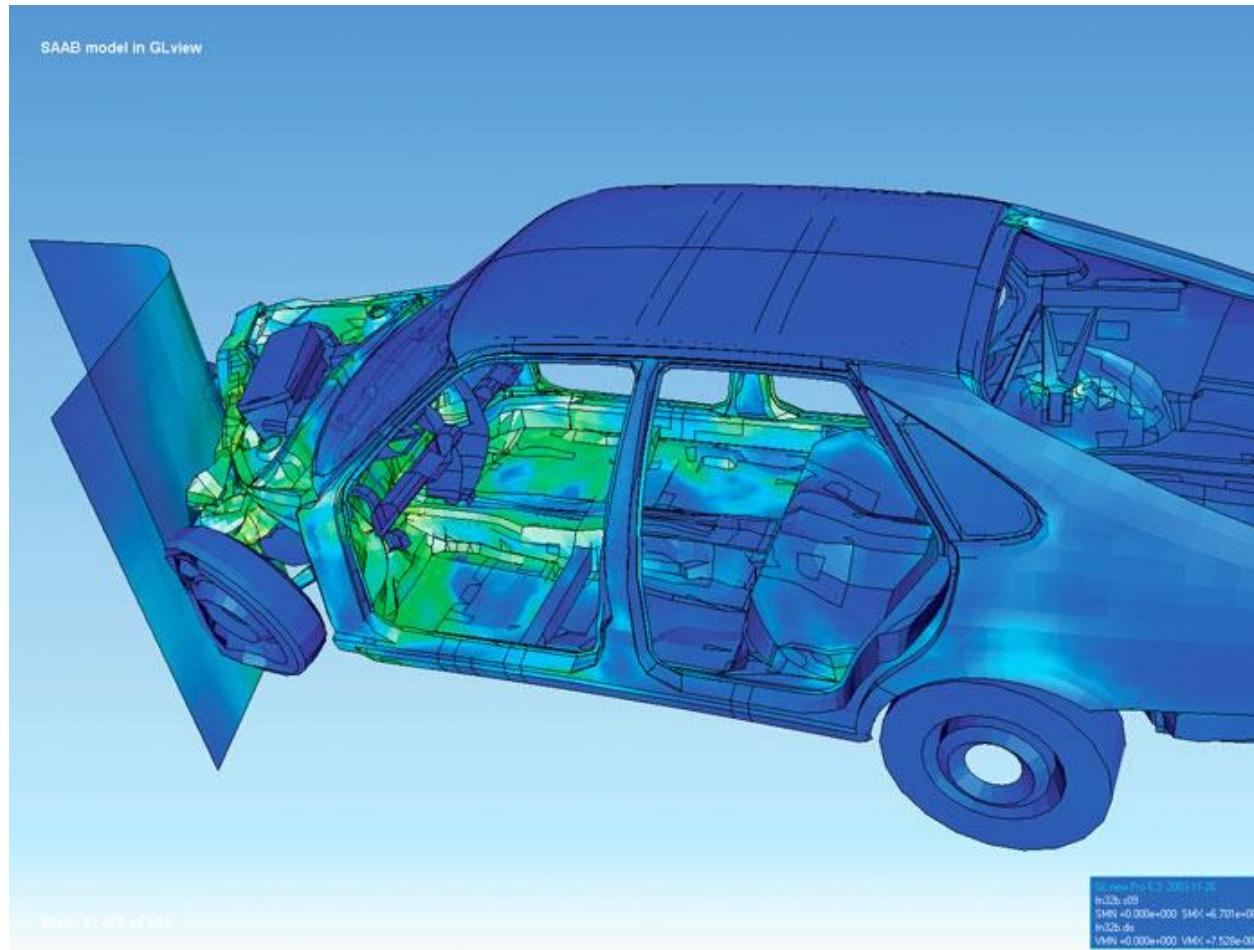
[Wikipedia]

Data Visualization



[Wikipedia]

CAD – Simulation and Visualization



[Wikipedia]

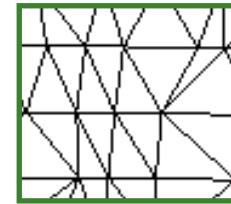
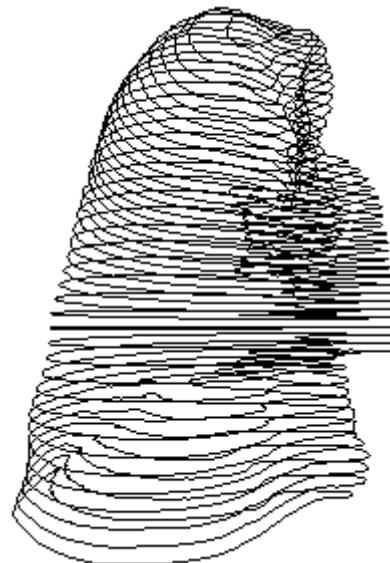
3D models – Applications

■ Virtual / Augmented reality

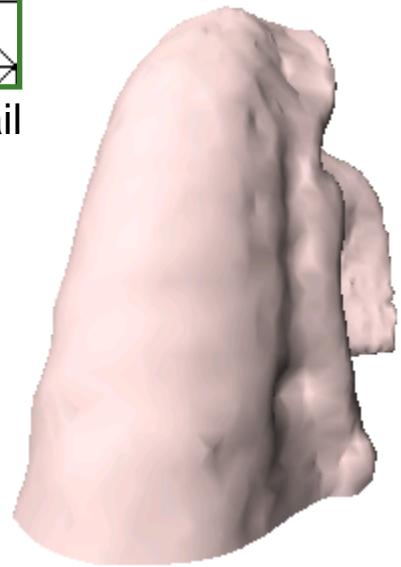


3D models – Applications

■ Medical Data Processing



Mesh detail



Mesh

3D models – Applications

■ Other application areas

- Computer games
- Geographical information systems (GIS)
- Engineering analysis
- 3D printing / Rapid prototyping
- Medical solid modeling
- ...

3D models – Shape

- Define **from scratch** using VRML / X3D, OpenGL, VTK, ...
 - Tedium; requires skill
- Obtain from **CAD files or model databases**
 - Convert to compatible formats
 - Use of existing models in manufacturing applications
- Create using a **3D digitizer** or a **3D scanner**
 - 3D digitizer : stylus
 - 3D scanner : tracker, cameras and laser

3D modeling tools

- Spatial Corp.'s **ACIS**; 3D modeling engine
 - <http://www.spatial.com>
- Siemens's **Parasolid**; 3D modeling engine
 - <http://www.plm.automation.siemens.com>
- Dassault Systemes's **CATIA**; CAD / CAM / CAE
 - <http://www.3ds.com>
- PTC's **Pro/ENGINEER**; 3D feature modeling
 - <http://www.ptc.com>
- **SolidWorks**; 3D feature modeling
 - <http://www.solidworks.com>

3D modeling tools

- Autodesk's **3ds max** and **Maya**
 - <http://www.autodesk.com>
- **Blender**: Free open-source 3D content-creation suite
 - <http://www.blender.org>
- **Rhino**: Uninhibited free-form 3D modeling
 - <http://www.rhino3d.com>
- **Trimble SketchUp**: Intuitive 3D modeler
 - <http://www.sketchup.com>
- **POV-Ray**: Persistence of Vision Ray-Tracer
 - <http://www.povray.org>

POLYGONAL MESHES

Geometric Modeling

■ Main areas

- Curve and surface modeling 

 - Computer-Aided Geometric Design (CAGD)

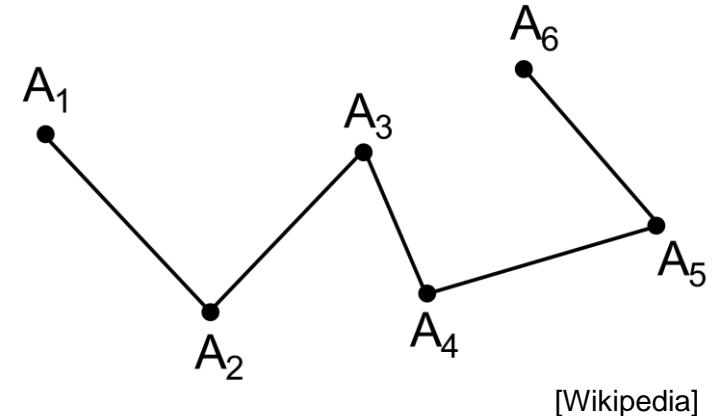
- Solid Modeling
- Volume Modeling

■ Simplest models

- Curves : Polygonal lines
- Surfaces : Polygonal meshes 

Polyline

- Questions :
 - Open or closed polylines ?
 - Exact representation ? When ?
 - Approximate representation ?
- A “good” approximation usually needs a larger number of points
 - Level of detail (**LOD**)
- Typical application
 - Representing contours in processed images, after locating dominant points



[Wikipedia]

Polyline



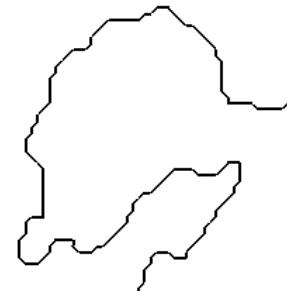
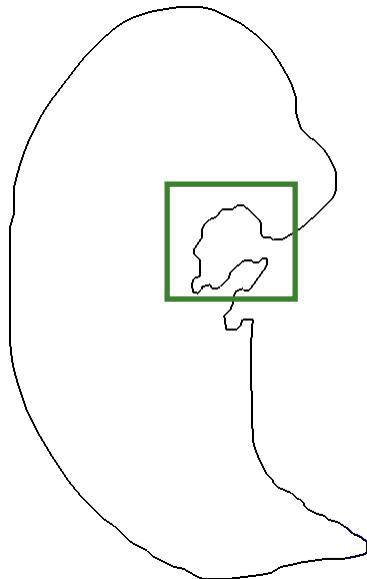
Contours segmented from CT images

Contour description is point by point

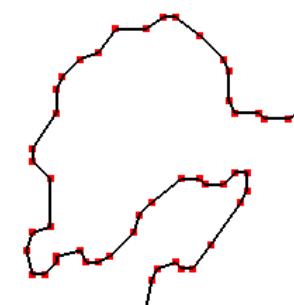


Too many points !

Solution: Dominant Point Detection



Contour
detail



Dominant points

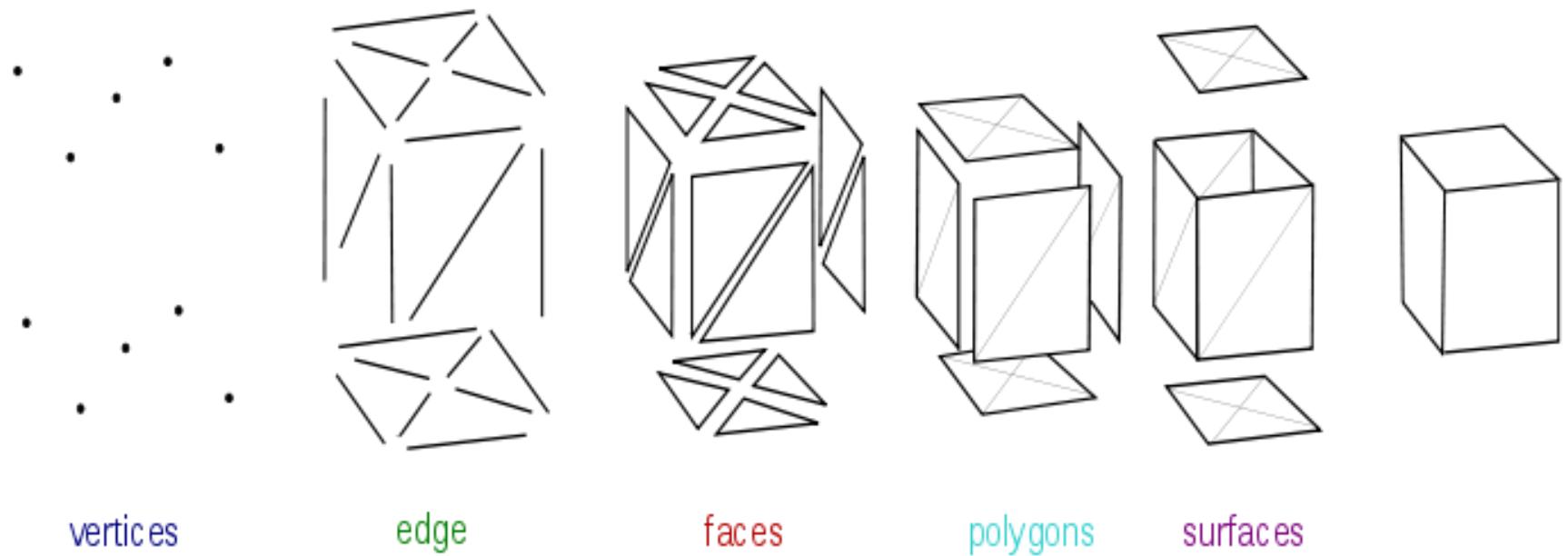


Polyline

Polygonal Meshes

- Surface is defined as a collection of neighboring faces (e.g., triangles)
 - **Geometry + Topology** (i.e., connectivity)
 - Vertices, edges, faces
- Euler formula for closed surfaces
 - $V + F - E = 2$
- Exact vs approximate representations
 - Polyhedral models
 - Curved surfaces
 - Terrain models
 - Complex surfaces / models

Polygonal Meshes



[Wikipedia]

Polygonal Meshes

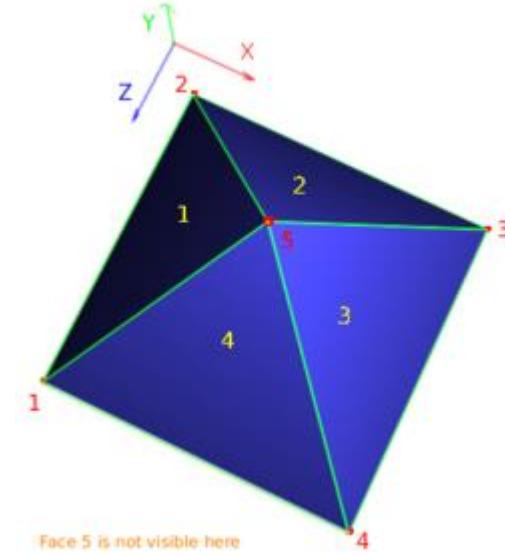
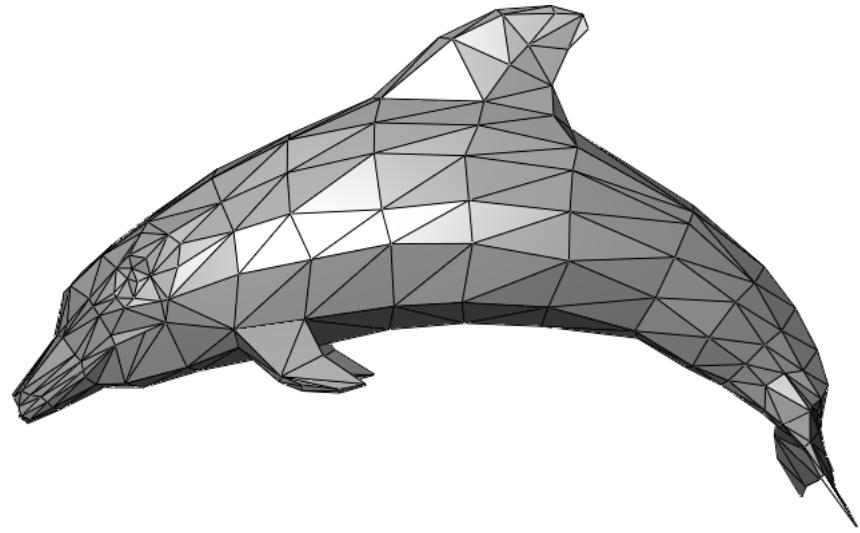
- Collection of neighboring **vertices**, **edges** and **polygons**
 - Usually **triangles** !!
- Vertex
 - Shared by, at least, 2 edges
- Edge
 - Connects 2 vertices
 - Shared by 2 polygons, if the surface is closed
- Polygon
 - Sequence of, at least, 3 vertices

Polygonal Meshes

- Homogeneous ?
- Adaptive ?
- Easy to render

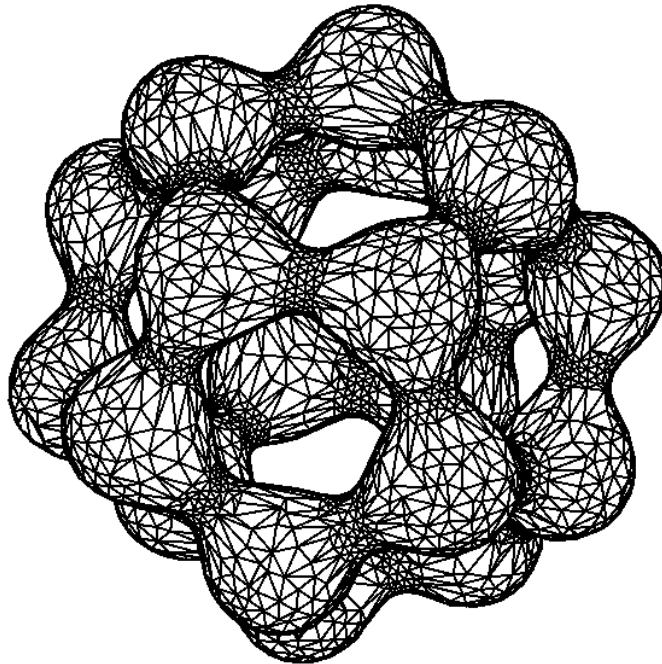
- Usually triangles !

- Pyramid
 - How many entities?
 - Check Euler formula!



[Wikipedia]

Polygonal Meshes



Complex topology



Complex geometry

[Seidel and Belyaev, 2006]

Polygonal Meshes

- What / How to store?
 - Memory or file?
 - List of vertices – Topology ?
 - List of triangles – Neighbors?
 - Lists of vertices, edges and triangles – Efficiency?
 - Winged-edge or half-edge data structure
- Common operations
 - Smoothing
 - Decimation
- Toolboxes / Libraries
 - CGAL
 - OpenMesh

Polygonal Meshes

- The **surface** (i.e., the model) is defined as a set of adjacent faces (e.g., **triangles**)
- Which **geometric information** should be stored ?
 - Vertex coordinates
- Which **topological information** (i.e., connectivity) should be stored ?
 - How are edges and faces arranged ?
 - How to identify **neighboring / incident /adjacent** entities ?
 - Efficiency !
- Which **additional properties** should be stored ?
 - Normal vector to each face / vertex
 - Texture coordinates
- How to check the **validity** of a model ?
 - 2-manifolds
 - Euler Formulae

Some basic operations

- Find the vertices defining an edge
- Find the edges incident in a vertex
- Find all polygons sharing
 - A vertex
 - An edge
- Identify mesh errors. I.e., the lack of
 - A vertex / an edge / a face
- Rendering a mesh

Polygonal Meshes

- Supported by most applications
- Various **file formats**
- **Triangle meshes** are the most common !!
 - Planar faces
 - Algorithm simplicity
 - Numerical robustness
 - Efficient rendering

Polygonal Meshes

- Exact vs. approximate rep. – When ?
 - Polyhedral models
 - Curved surfaces
 - Terrain models
 - More complex models / surfaces
- A “good” approximation might require a large number of faces
 - Levels-of-Detail (**LODs**)

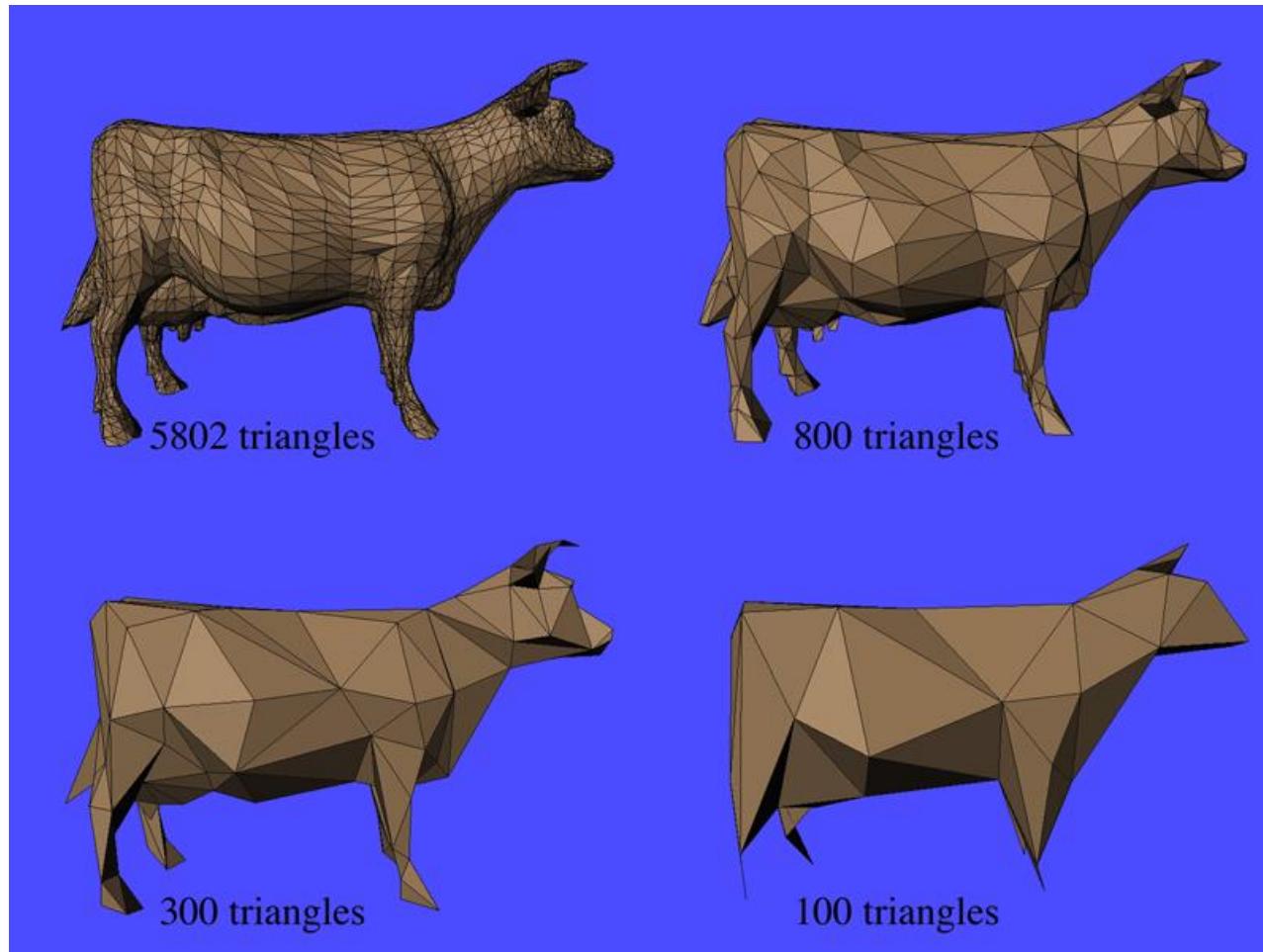
Polyhedral models

- The same polyhedral model might be represented by **different polygonal meshes** !!
 - Useful for shading / rendering
- Degrees of freedom
 - **Number** of mesh vertices
 - **Distribution** of mesh vertices
 - **Arrangement** of edges / polygons
- Example
 - Represent a cube using different polygonal meshes

Curved surfaces

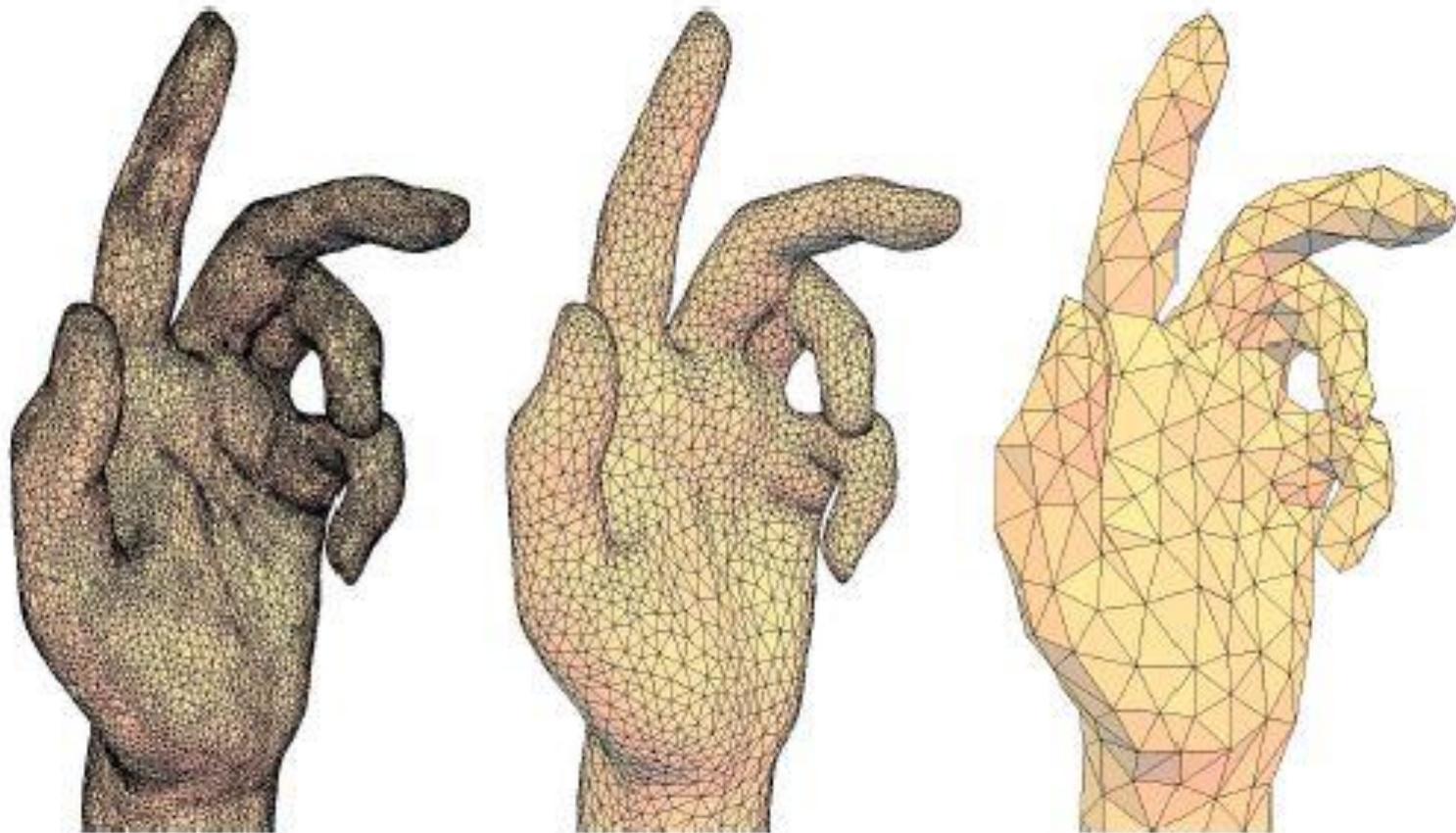
- Representing the shape of a curved surface is an **approximation** process
- There is no “unique” model!!
- Degrees of freedom
 - **Number** of mesh vertices
 - **Distribution** of mesh vertices
 - **Arrangement** of edges / polygons

How many triangles should be used?



[CMU, 2000]

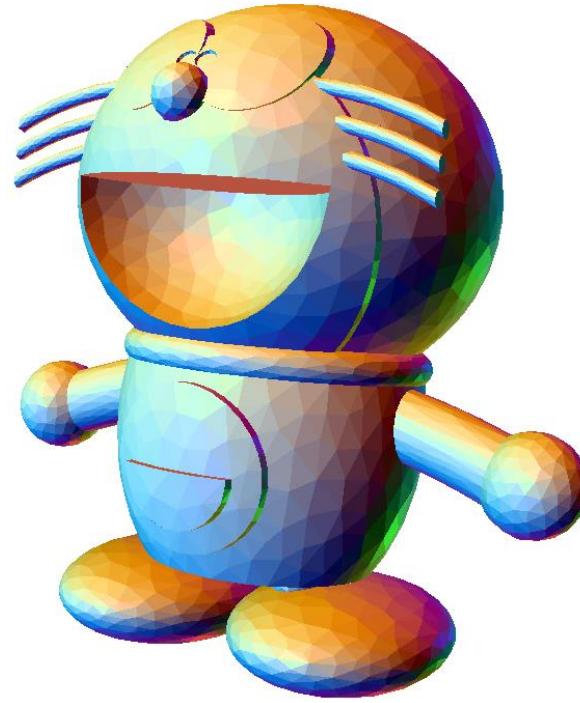
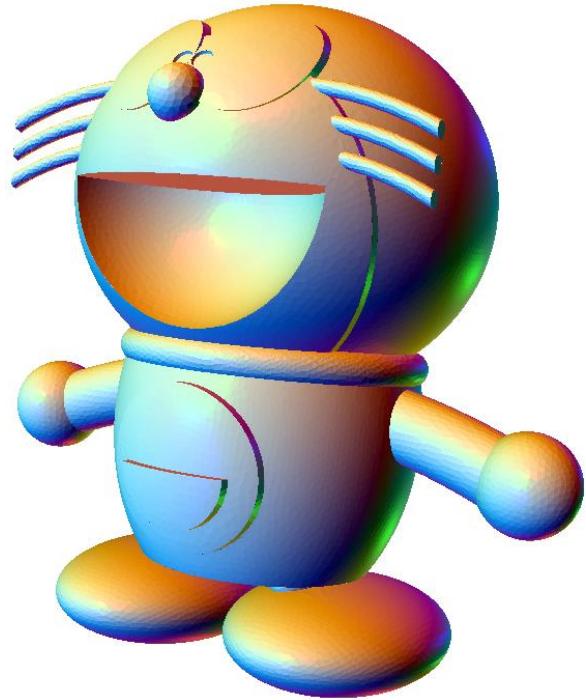
How many vertices should be used?



(a) 25,000 vertices. (b) 5,000 vertices. (c) 500 vertices.

[Dyer et al.]

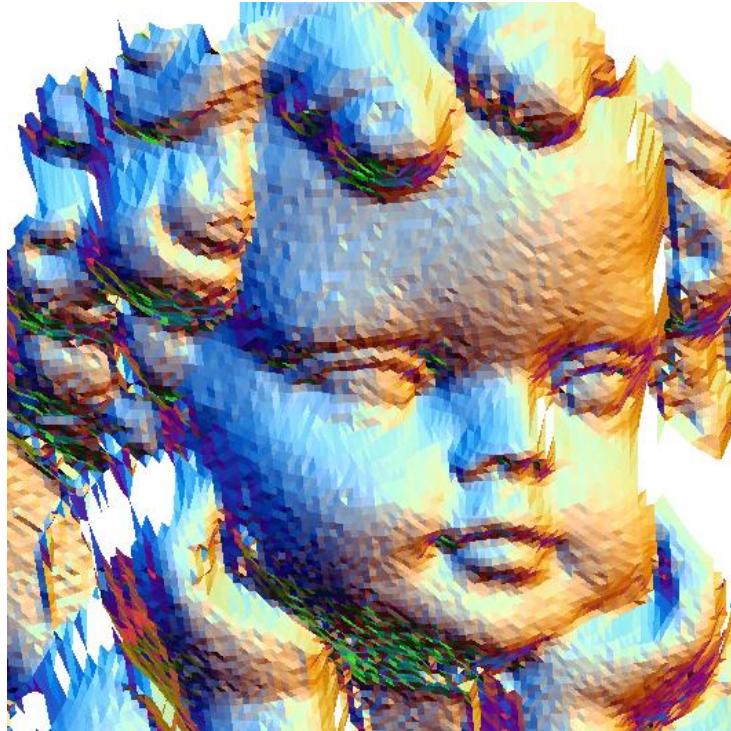
Mesh decimation



**90%
reduction**

[Seidel and Belyaev, 2006]

Mesh smoothing



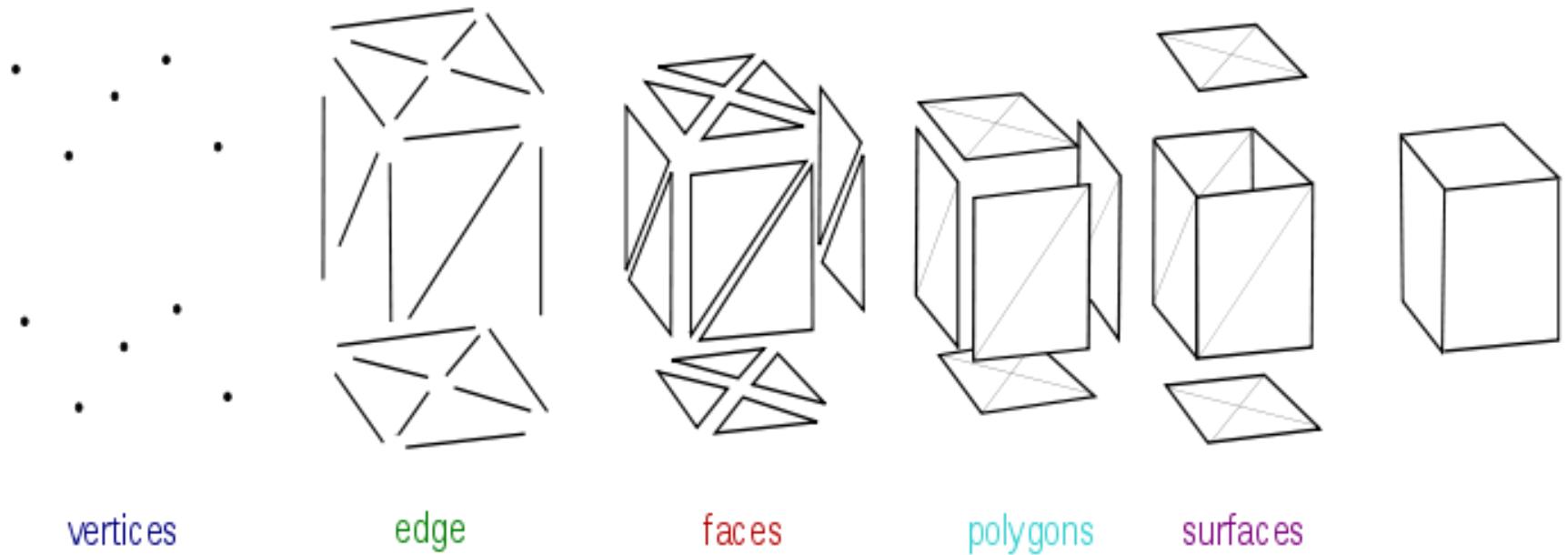
[Seidel and Belyaev, 2006]

GEOMETRY + TOPOLOGY

Representing model surfaces

- Geometrical information
 - Vertex coordinates
- Topological or connectivity information
 - Abstract definition of vertices, edges and faces
 - Incidence and adjacency information
- Properties
 - Normal vectors (“Normal Maps”)
 - Texture coordinates

Topological information



[Wikipedia]

Topological information

- Vertex
 - Regular ?
 - Singular ?
- Edge
 - 2 vertices
 - Border edge : just 1 incident face
 - Regular edge : 2 incident faces
 - Singular edge : 3 or more incident faces
- Loop
 - Ordered edge sequence

Topological information

- **Face**

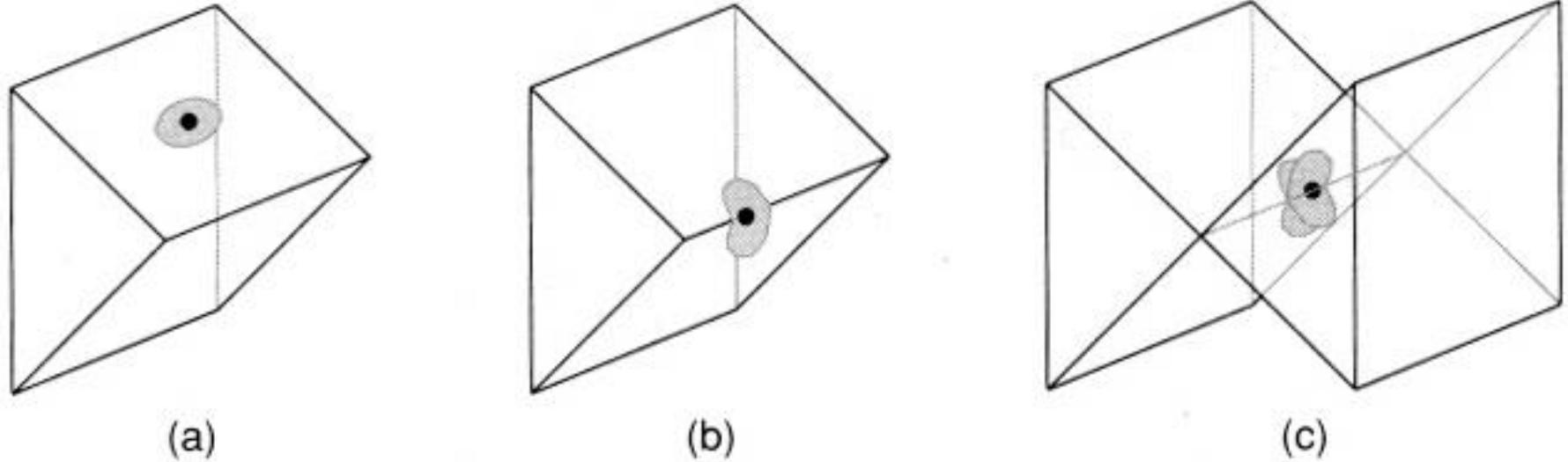
- Limited by a set of disjoint edge sequences
 - Outer border
 - Possible inner borders (“holes”)

- **Shell**

- Set of connected faces

- Examples ?

Valid vs. non-valid models



[Foley et al.]

Valid vs. non-valid models

- 2-Manifold Model
 - Any point has a “disk” neighborhood
 - No singular vertices !!
 - No singular edges!!
- Non-Manifold Model
 - **Dangling** Edges / Faces
 - **Touching** Faces
 - ...
 - **Non-valid** / non-manufacturable models !!
- Examples ?

Euler Formula

- Allows checking the consistency of the topological information !!
- $V + F - E = 2$
- When to apply?
 - Model has a closed, orientable surface !
 - Each face is limited by a single edge loop !
 - No through-holes !
 - Nor cavities !
- Examples
 - Tetrahedron
 - Different mesh representations of a cube

Euler-Poincaré Formula

- Generalization !!
- $V + F - E - (L - F) - 2(S - G) = 0$
- L – Number of loops
- S – Number of shells
- G – Genus : number of “handles”
- When to apply ?
 - Through-holes
 - Cavities
- Example ?

Consistency checking

- Check if
 - All polygons have closed borders
 - All edges are used at least once
 - Every vertex belongs at least to
 - 2 edges
 - 1 polygon
 - ...

COMPUTATIONAL REPRESENTATION

Computational representation

- Memory or file ?
- Vertices list
 - Topological information ??
- Polygons list / Detached triangles
 - How to identify neighbors ??
- Vertices, edges and polygons lists
 - Efficiency ?
- Winged-edge or half-edge data structures

Detached / Isolated Polygons List

- Each polygon is represented by the ordered list of its vertices coordinates
 - CCW
- Inefficient !!
 - Memory space : multiple vertex representation
 - Lack of information about shared vertices / edges
 - Cumbersome detection !!
 - Rendering : edges are drawn twice !!
- Example ?

Vertices List

- Vertices list / array
 - Store just once the coordinates of each vertex !
 - Easy to edit / modify one vertex
- Each polygon is described by its vertices sequence
 - Pointer / index
 - Usage : storing in a file
- Inefficient !!
 - Hard to detect which polygons share a given edge !!
 - Rendering : edges are drawn twice !!
- Example ?

Indexed Face Set

- VRML or MCGL or ...
- Array 3D vertex coordinates
 - One index for each vertex
- Convex n-sided polygons defined by n indices
- Example
 - [0,1,2,-1,2,1,3,4,-1]

OBJ File Format

■ Vertices list

v 10 15 20

v 23 34 56

...

■ Faces list

f 1 2 3

f 2 3 4

...

■ Additional information

- Normal vectors
- Texture coordinates
- ...

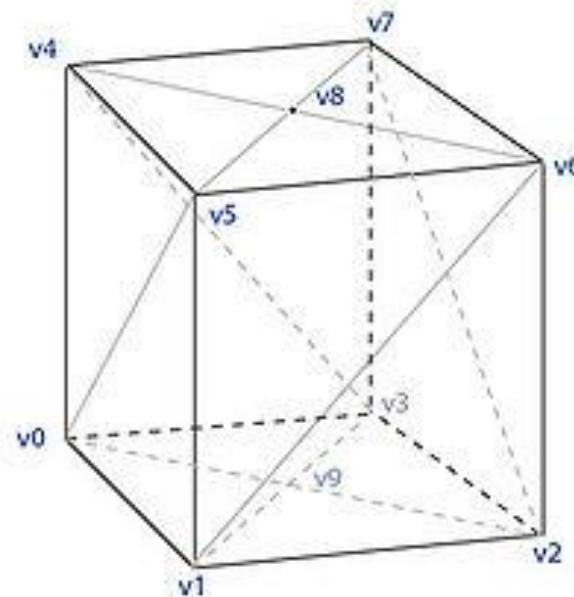
Topological information

- How to store incidence and adjacency information ?
- How to answer **basic queries** fast ?
 - Which are the **end vertices** of a given edge ?
 - Which are the **adjacent polygons** of a given edge ?
 - Which are the **incident edges** in a given edge ?
 - Which are the **incident edges** in a given vertex?
 - Which are the **neighboring vertices** of a given vertex ?
 - ...
- Efficiency
 - Time ?
 - Space ?

Adjacent Vertices List

Vertex-Vertex Meshes (VV)

Vertex List	
v0	0,0,0
v1	1,0,0
v2	1,1,0
v3	0,1,0
v4	0,0,1
v5	1,0,1
v6	1,1,1
v7	0,1,1
v8	.5,.5,0
v9	.5,.5,1



[Wikipedia]

Vertices List + Faces List

Face-Vertex Meshes

Face List	Vertex List
f0	v0 v4 v5
f1	v0 v5 v1
f2	v1 v5 v6
f3	v1 v6 v2
f4	v2 v6 v7
f5	v2 v7 v3
f6	v3 v7 v4
f7	v3 v4 v0
f8	v8 v5 v4
f9	v8 v6 v5
f10	v8 v7 v6
f11	v8 v4 v7
f12	v9 v5 v4
f13	v9 v6 v5
f14	v9 v7 v6
f15	v9 v4 v7

v0	0,0,0	f0 f1 f12 f15 f7
v1	1,0,0	f2 f3 f13 f12 f1
v2	1,1,0	f4 f5 f14 f13 f3
v3	0,1,0	f6 f7 f15 f14 f5
v4	0,0,1	f6 f7 f0 f8 f11
v5	1,0,1	f0 f1 f2 f9 f8
v6	1,1,1	f2 f3 f4 f10 f9
v7	0,1,1	f4 f5 f6 f11 f10
v8	.5,.5,0	f8 f9 f10 f11
v9	.5,.5,1	f12 f13 f14 f15

example →

The diagram shows a cube with vertices labeled v0 through v9. The vertices are arranged as follows: v0 (bottom-front), v1 (bottom-back), v2 (top-back), v3 (top-front), v4 (top-right), v5 (top-left), v6 (bottom-right), v7 (bottom-left), and v8 (top-center). The faces are labeled f0 through f15. f0 is the front face (v0-v1-v2-v3). f1 is the back face (v4-v5-v6-v7). f2 is the bottom face (v0-v1-v6-v5). f3 is the top face (v4-v7-v8-v9). f4 is the right face (v2-v3-v8-v7). f5 is the left face (v0-v1-v5-v4). f6 is the back-right face (v5-v6-v7-v8). f7 is the back-left face (v4-v5-v9-v8). f8 is the bottom-right face (v1-v6-v8-v9). f9 is the bottom-left face (v0-v1-v4-v5). f10 is the front-right face (v2-v3-v7-v6). f11 is the front-left face (v0-v1-v4-v3). f12 is the top-right face (v7-v8-v9-v6). f13 is the top-left face (v4-v7-v8-v5). f14 is the back-right-top face (v7-v8-v9-v5). f15 is the back-left-top face (v4-v5-v9-v6).

[Wikipedia]

Textures

Joaquim Madeira

May 2022

Overview

- Motivation
- Textures
- Texture Mapping
- Textures in Three.js

MOTIVATION

Geometric Modeling – Limits

- Graphics cards can render **millions of triangles per second**
- **BUT**, that might not be sufficient...
- Skin / Terrain / Grass / Clouds / ...

How to model / render an orange ?

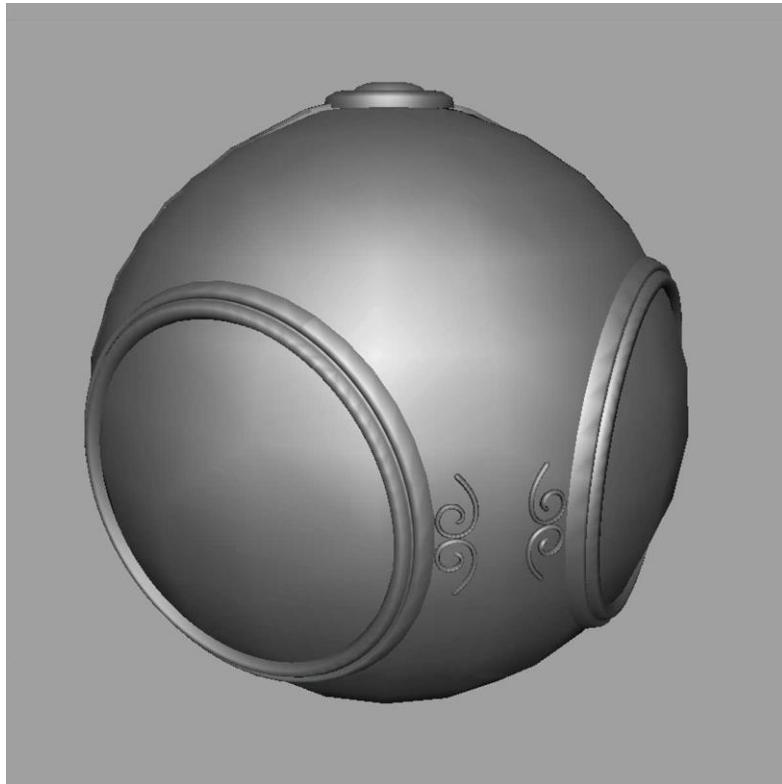
- An orange colored **sphere** ?
 - Too simple !
- A more **complex** shape to convey **details** ?
 - How to represent **surface features** ?
 - Takes **too many triangles** to model all the dimples...

How to model / render an orange ?

- Simple geometric model + Texture
 - Take a picture of a real orange
 - Scan and “paste” it onto model
 - Texture mapping
- Might not be sufficient: surface will be smooth
- How to “change” local shape ?
 - Bump mapping

TEXTURES

Texture Mapping



geometric model

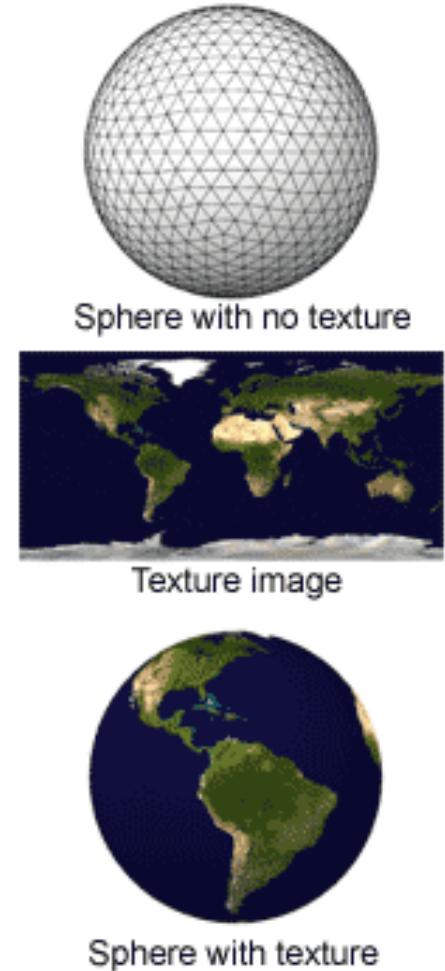


texture mapped

[Ed Angel]

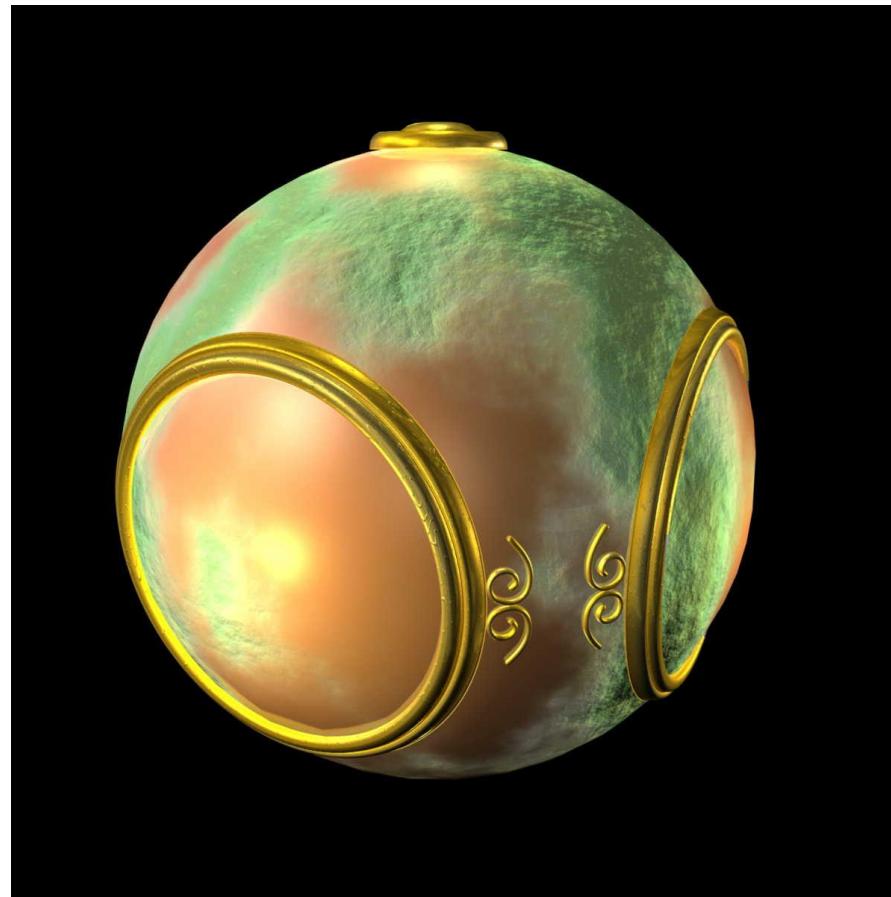
Texture Mapping

- Implemented in hardware on every **GPU**
- Simplest surface detail hack
- Paste the texture on a surface to **add detail** without adding more triangles
 - Get surface color or alter computed surface color



[Andy Van Dam]

Bump Mapping



[Ed Angel]

Environment Mapping



[Ed Angel]

Textures – Simulating Ray-Tracing



[<http://www.okino.com>]

- Increased realism !!
 - 11 light sources + 25 texture maps

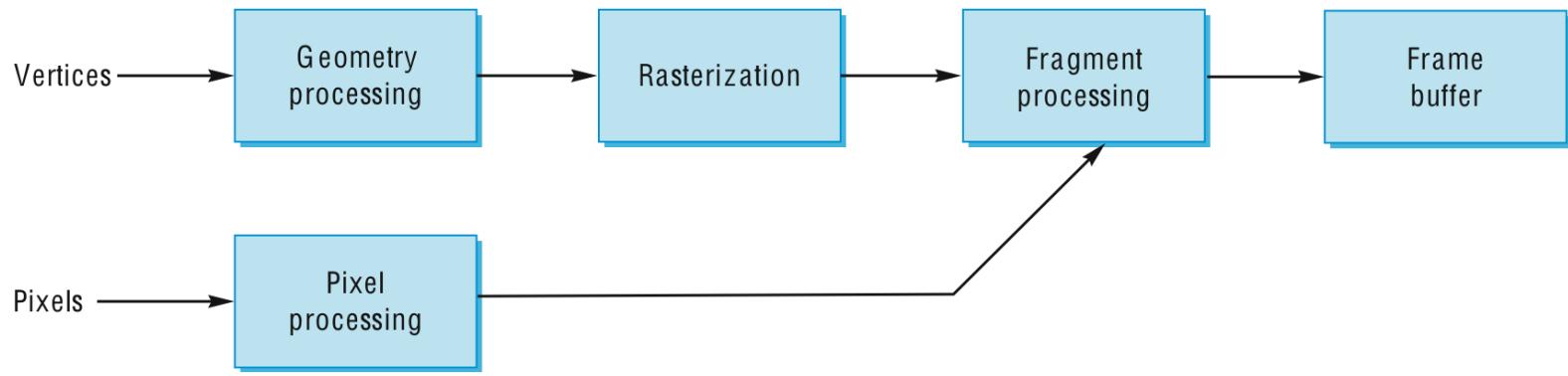
TEXTURE MAPPING

Mapping

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

Where does it take place ?

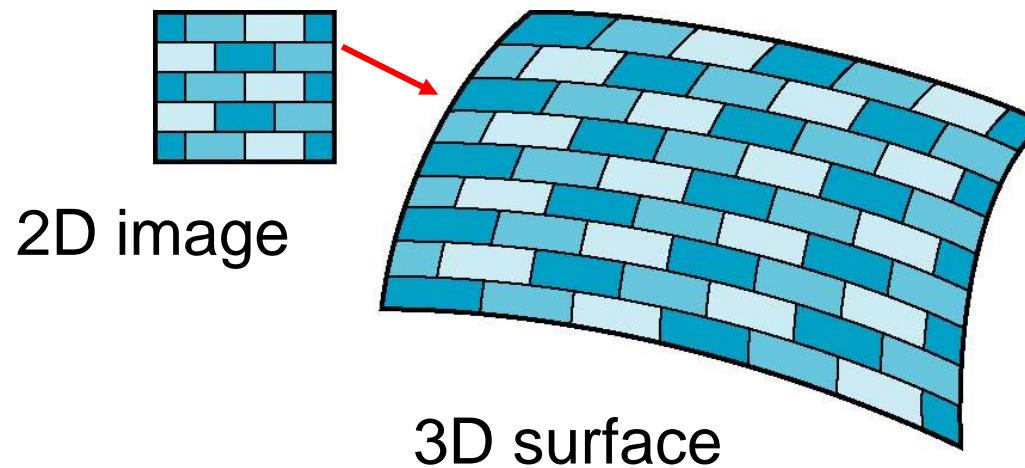
- Mapping techniques are implemented at the **end** of the rendering **pipeline**
 - Very efficient because **few polygons** make it past the clipper



[Ed Angel]

Mapping – Is it simple ?

- Although the idea is simple – map an image to a surface – there are 3 or 4 coordinate systems involved

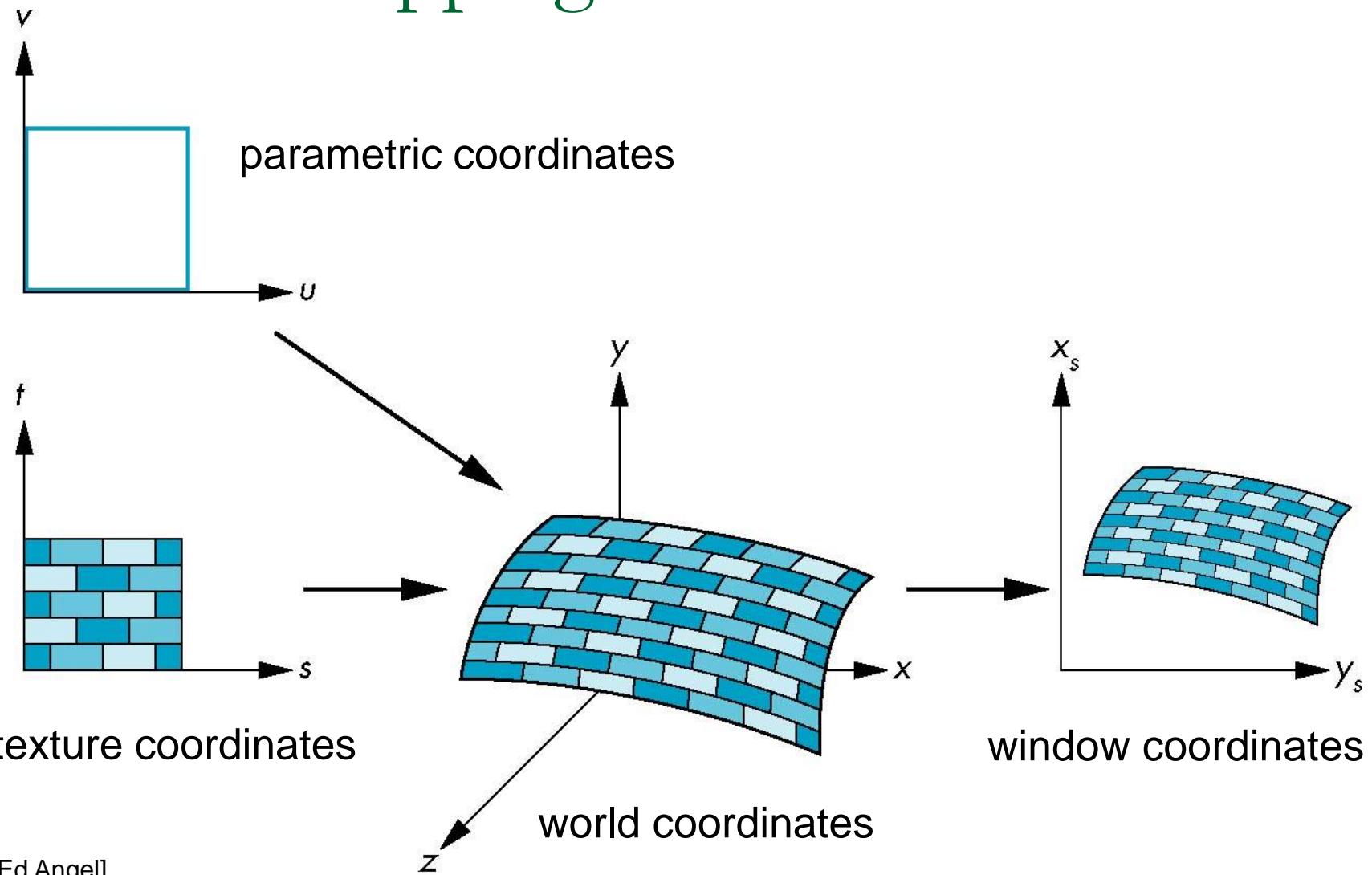


[Ed Angel]

Coordinate Systems

- Parametric coordinates
 - May be used to model **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

Texture Mapping



[Ed Angel]

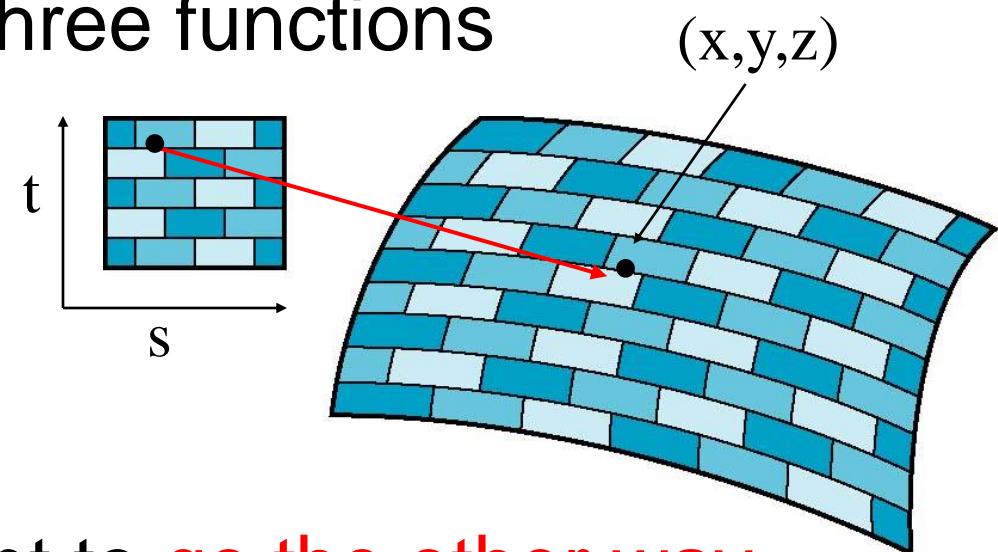
Mapping Functions

- Mapping from texture coordinates to a point on 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**

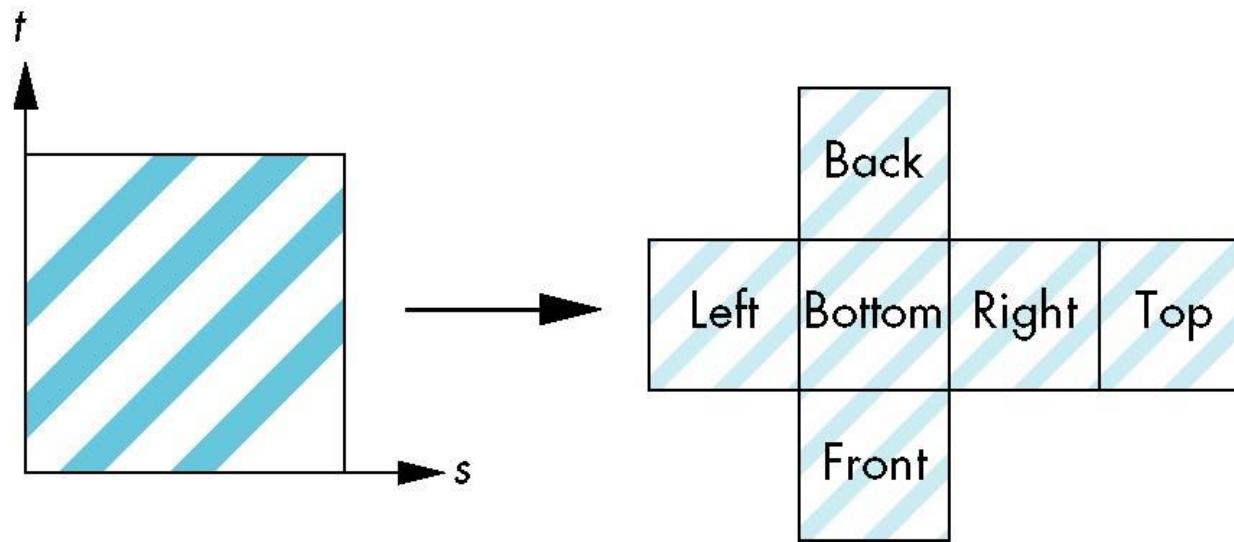
[Ed Angel]

Backward Mapping

- 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
 - $s = s(x, y, z)$
 - $t = t(x, y, z)$
- Such functions are difficult to find in general

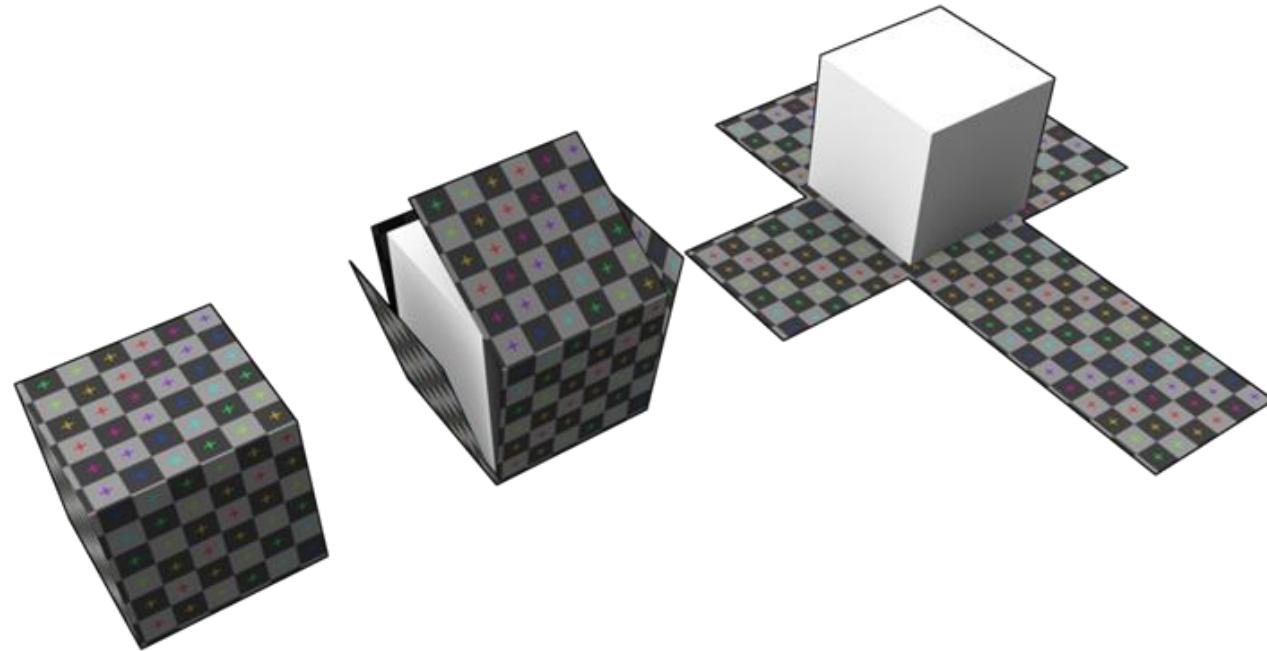
Box Mapping

- Easy to use with simple orthographic projection
- Also used in environment maps



[Ed Angel]

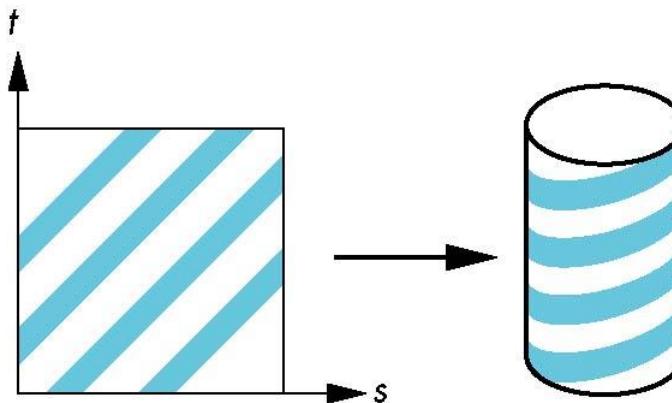
Example



[Andy Van Dam]

Two-part mapping

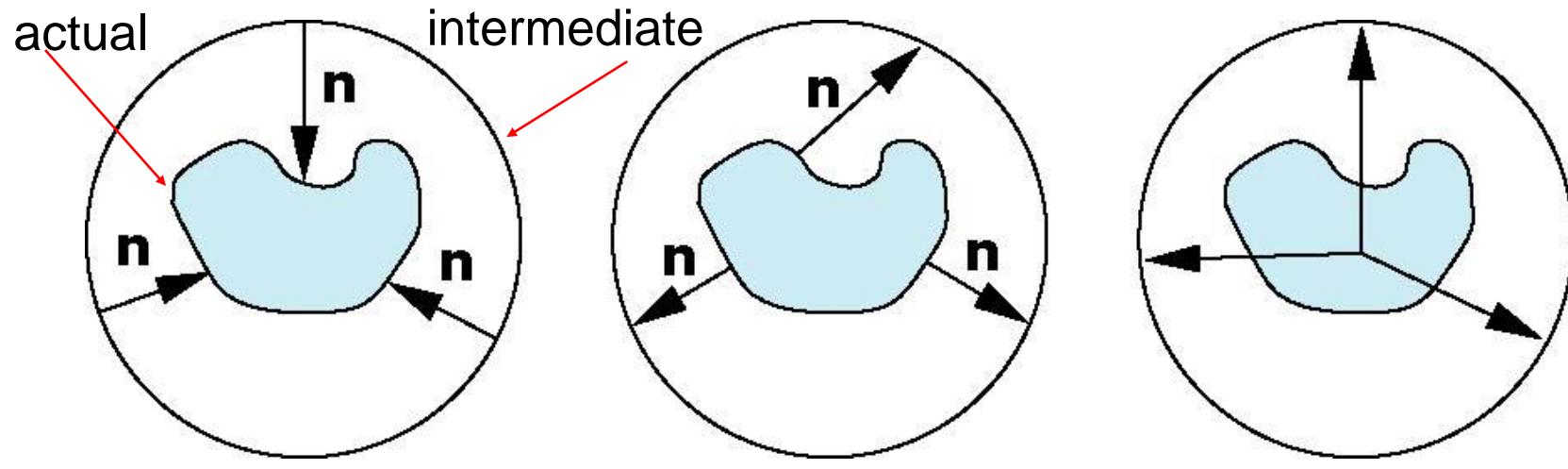
- One solution to the mapping problem is to **first** map the texture to a simple **intermediate surface**
- Example: map to cylinder



[Ed Angel]

Second Mapping

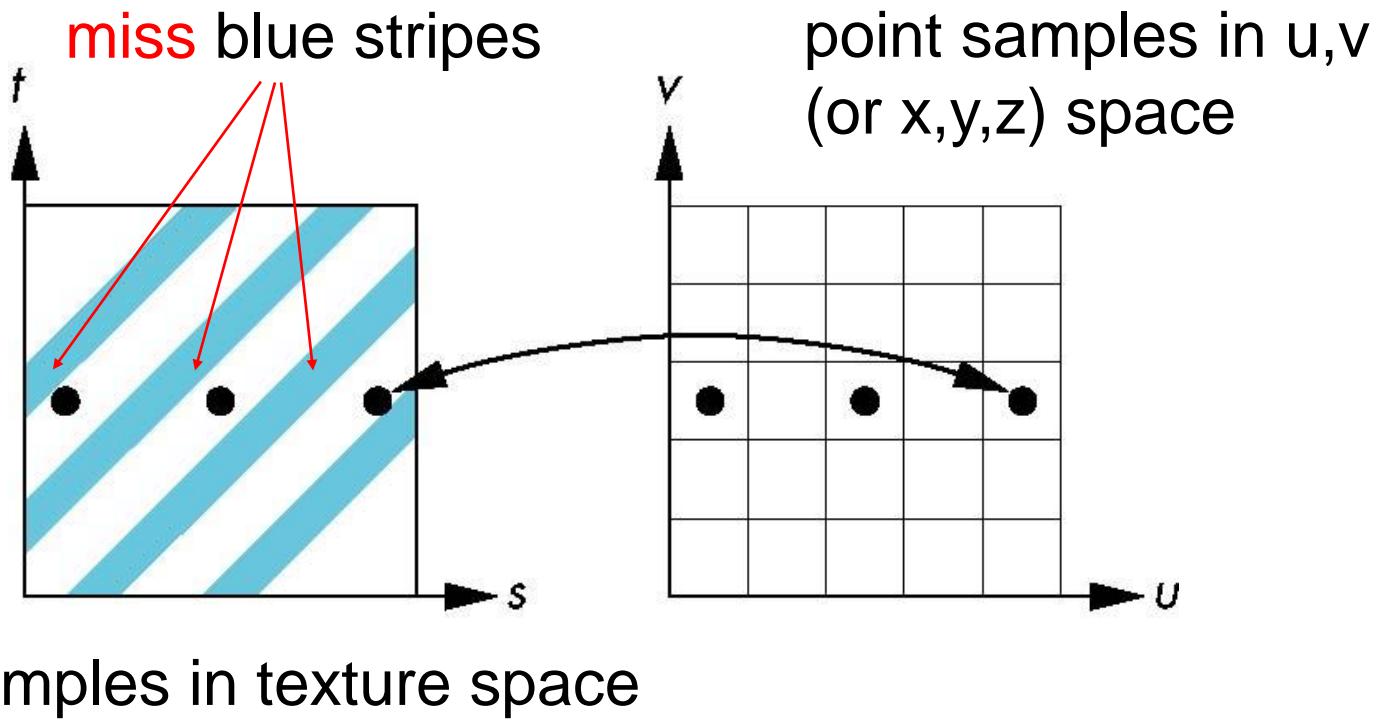
- Map from intermediate object to actual object
 - Normals from intermediate to actual
 - Normals from actual to intermediate
 - Vectors from center of intermediate



[Ed Angel]

Aliasing

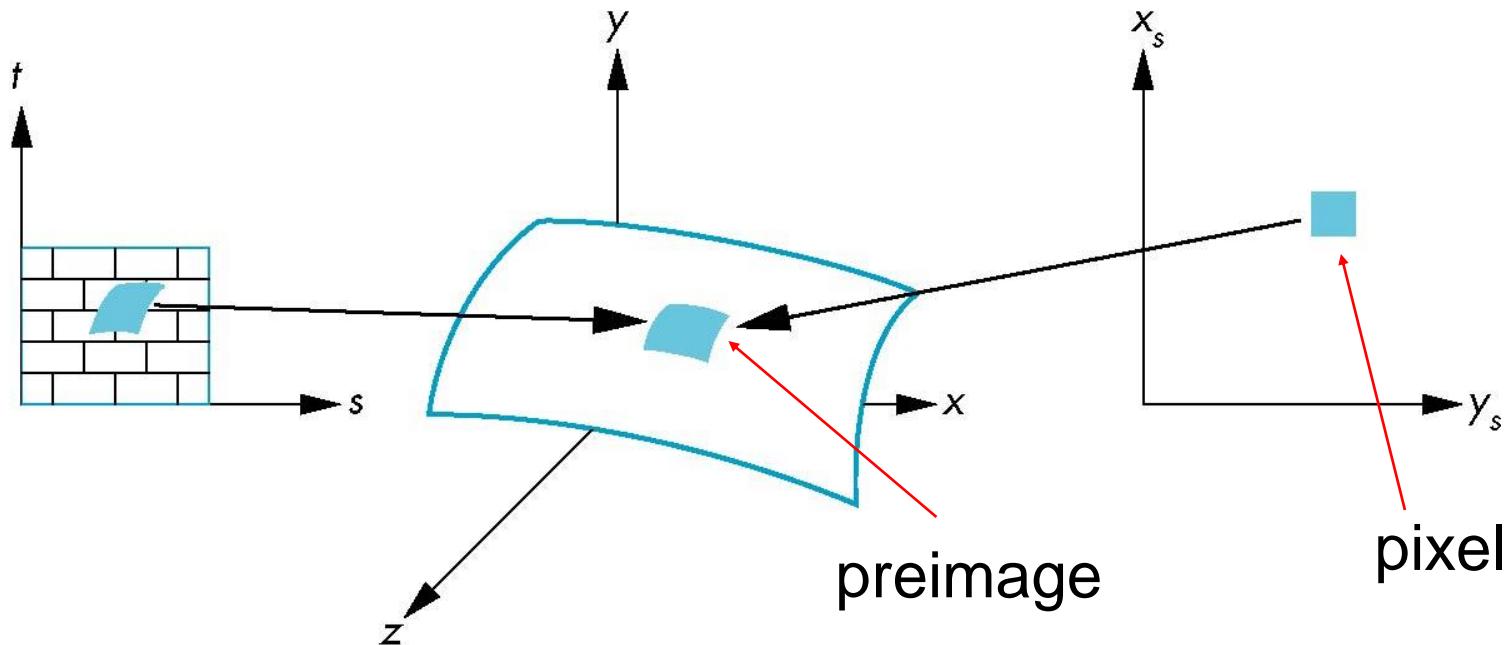
- Point sampling of the texture can lead to **aliasing errors**



[Ed Angel]

Area Averaging

- A better but slower option is to use *area averaging*

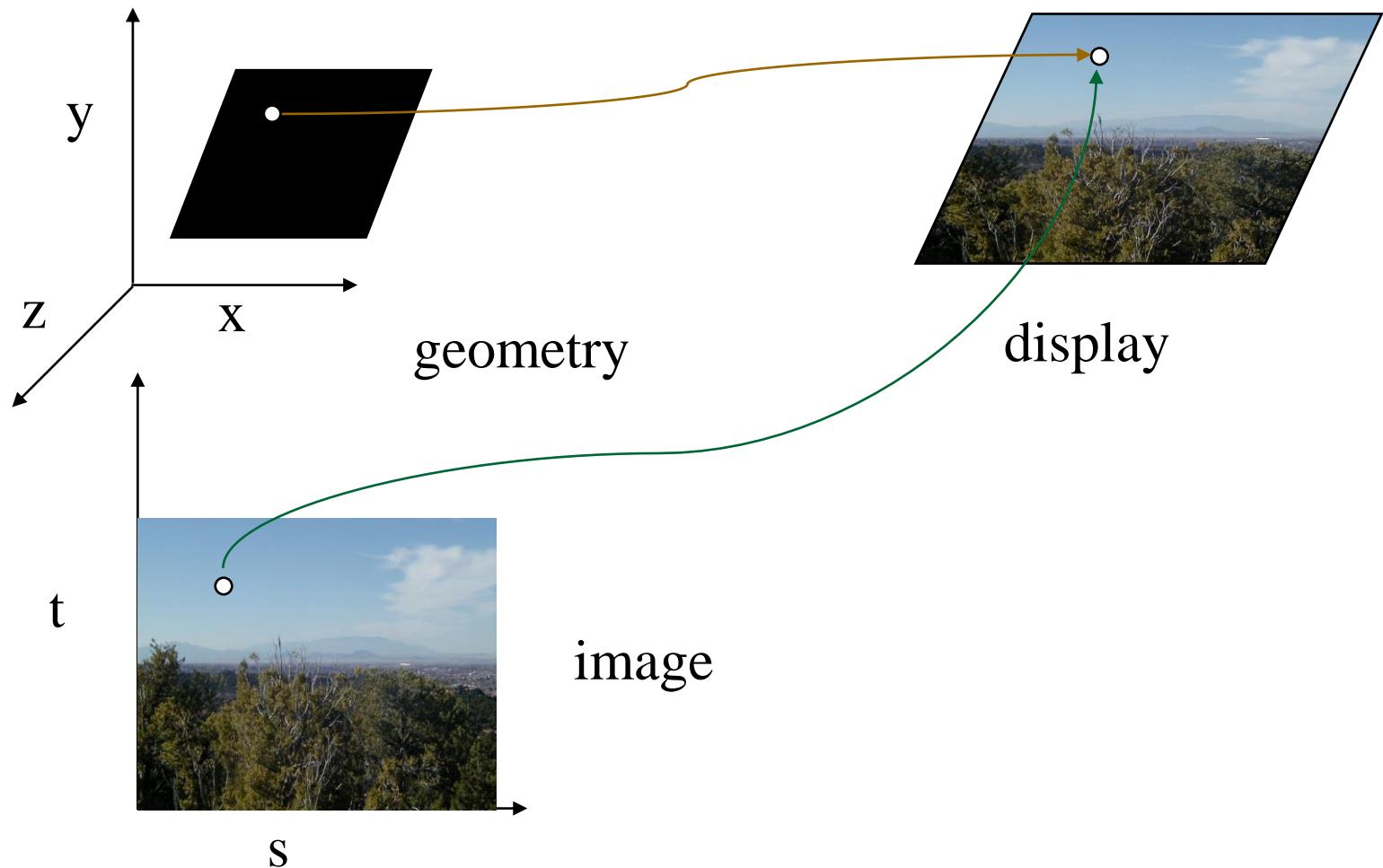


[Ed Angel]

Textures – Basic Strategy

- Three steps to applying a texture
 - specify the **texture**
 - read or generate image
 - assign to texture
 - enable texturing
 - assign **texture coordinates** to vertices
 - Proper **mapping** function is left to **application !!**
 - specify **texture parameters**
 - wrapping, filtering

Texture Mapping



[Ed Angel]

Texture Example

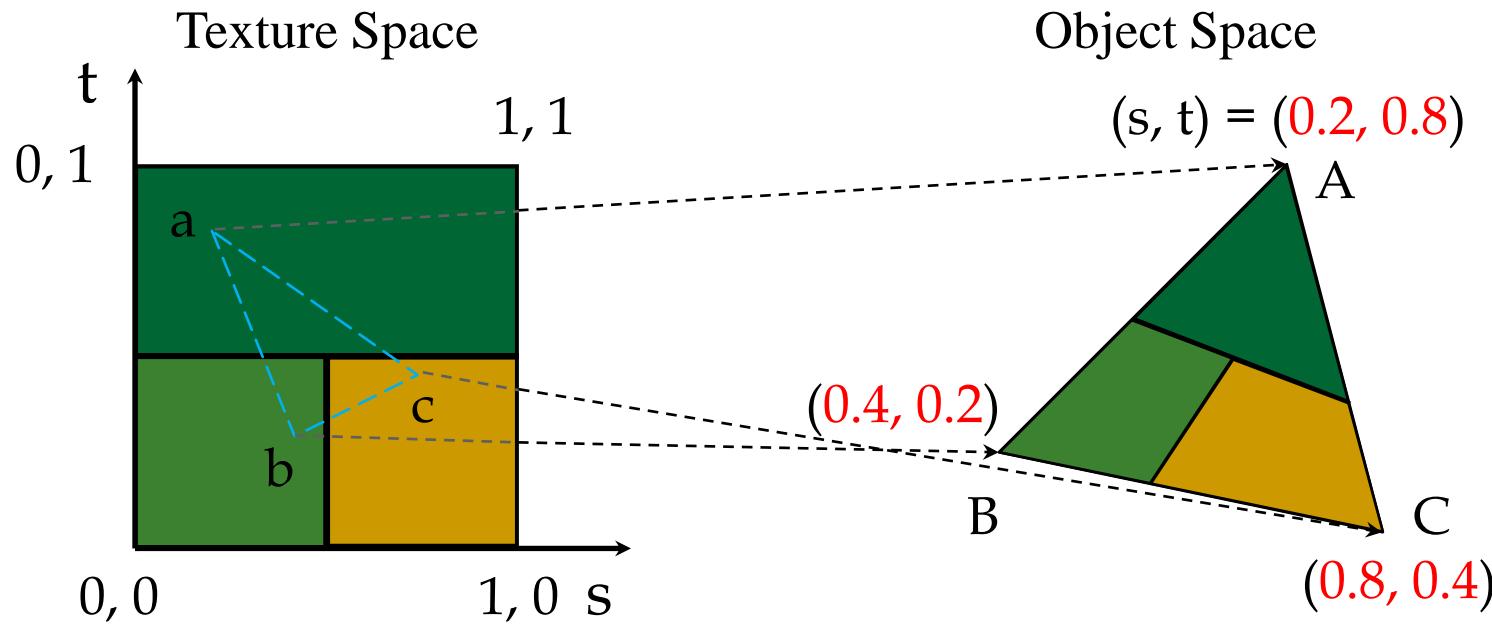
- The texture (below) is a 256 x 256 image
- It has been mapped to a rectangular polygon which is viewed in perspective



[Ed Angel]

Mapping a Texture

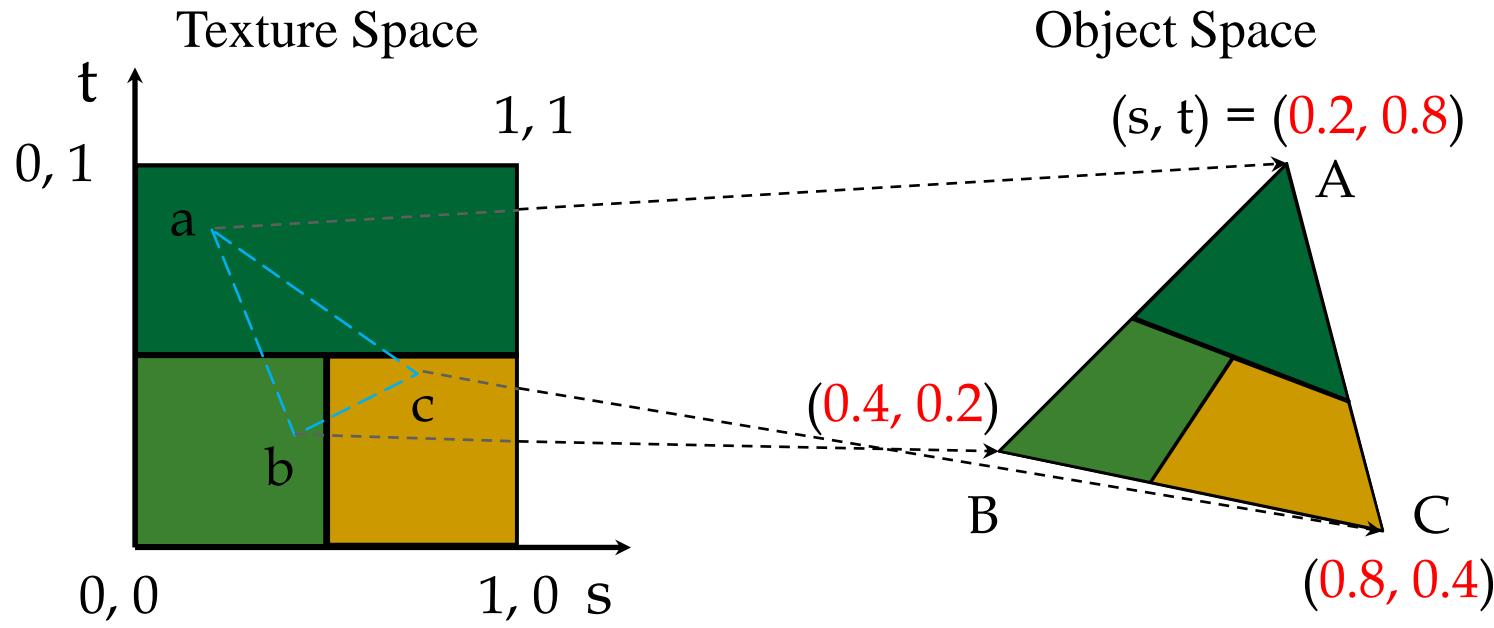
- Specify **texture coordinates** as a 2D **vertex attribute**
- Same vertex may have **different texture coordinates** for **different triangles**



[Ed Angel]

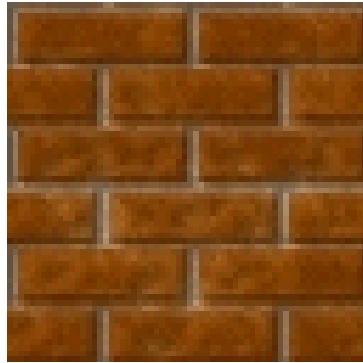
Mapping a Texture

- Texture coordinates are **linearly interpolated** across triangles

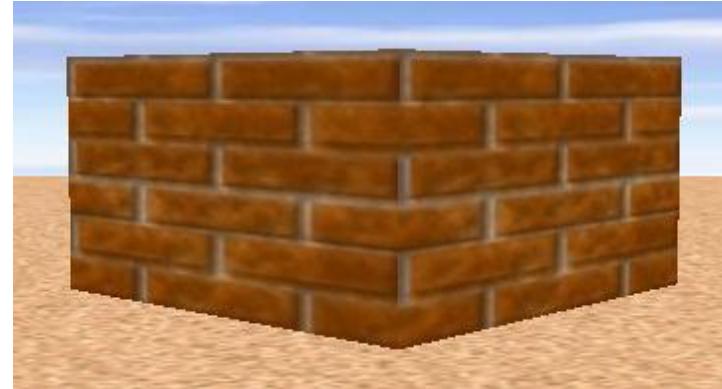


[Ed Angel]

Texture Mapping Style – Tiling



Texture



Without Tiling



With Tiling

[Andy Van Dam]

Texture Mapping Style – Stretching



Texture



Applied with stretching

[Andy Van Dam]

Texture Parameters

■ How is a texture applied ?

- **Wrapping** parameters determine what happens if s and t are outside the (0,1) range
- **Filter modes** allow us to use area averaging instead of point samples
- **Mipmapping** allows us to use textures at multiple resolutions
- **Environment parameters** determine how texture mapping interacts with **shading**

Other Texture Features

■ Environment Maps

- Start with image of environment through a wide-angle lens
 - Can be either a real scanned image
- Use this texture to generate a **spherical map**
- Alternative is to use a **cube map**

■ Multitexturing

- Apply a **sequence of textures** through cascaded texture units

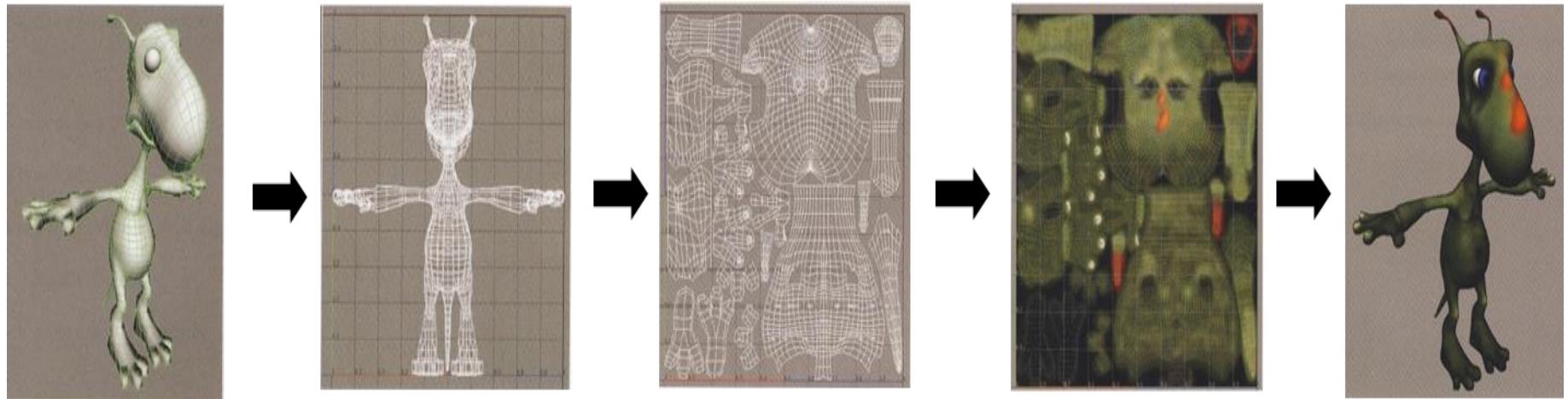
Applying Textures

- Textures can be applied in many ways
- A texture **fully determines** color
- A texture is **modulated** with a computed **color**
- A texture is blended with an **environmental** color

Complex Geometry/Real Applications

- Texture mapping of **complicated objects**, not simple primitives
- Need **precise control** over how the texture map looks on the object
- Use 3D modeling programs
 - E.g., **Maya**, Zbrush, Blender, ...

Complex Geometry/Real Applications



[Andy Van Dam]

TEXTURES

IN THREE.JS

What can we do ?

- Use a texture to define the **colors** of individual mesh pixels



[Dirksen]

Loading and applying a texture

```
function createMesh(geom, imageFile) {  
  
    var texture = THREE.ImageUtils.loadTexture(  
        "../assets/textures/general/" + imageFile );  
  
    var mat = new THREE.MeshPhongMaterial();  
    mat.map = texture;  
  
    var mesh = new THREE.Mesh(geom, mat);  
    return mesh;  
}
```

Loading and applying a texture

- Textures are correctly applied
- Almost any image can be used as a texture
- **Best results** are obtained with square images of size $2^k \times 2^k$
- **Loaders** for common formats

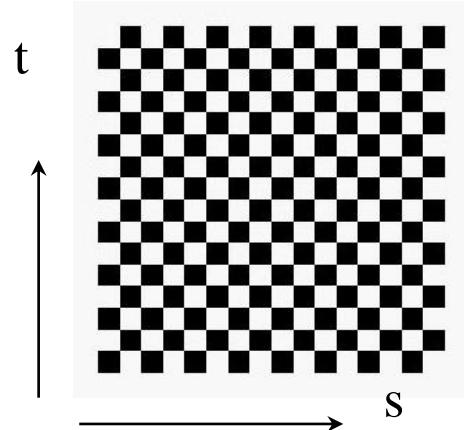


[Dirksen]

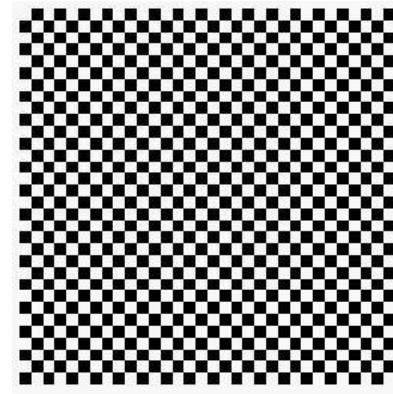
Loading and applying a texture

- Textures are correctly applied
 - For a **cube**, each side will show the complete image
 - For a **sphere**, the complete texture is wrapped around the sphere
- What if we want the texture to **repeat** itself ?

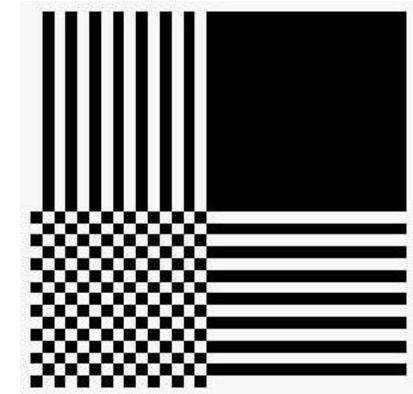
Wrapping Mode



texture



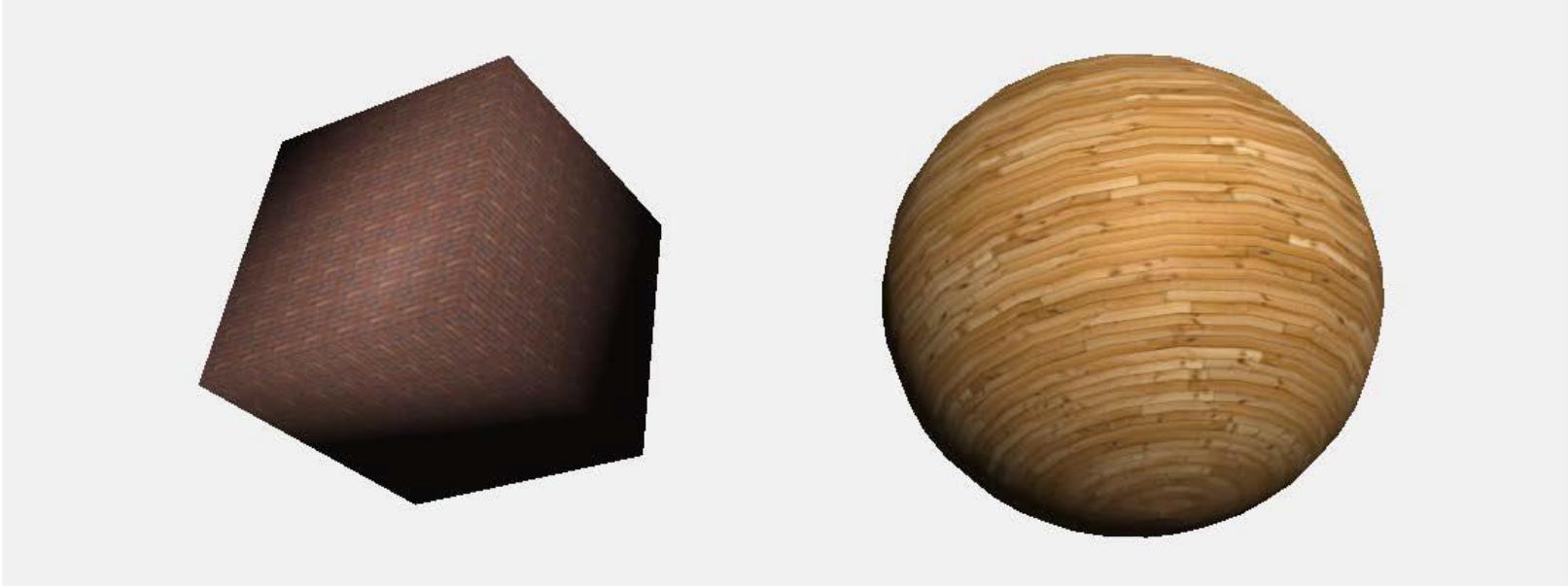
REPEAT
wrapping



CLAMP
wrapping

[Ed Angel]

Repeat Wrapping



```
cube.material.map.wrapS = THREE.RepeatWrapping;  
cube.material.map.wrapT = THREE.RepeatWrapping;
```

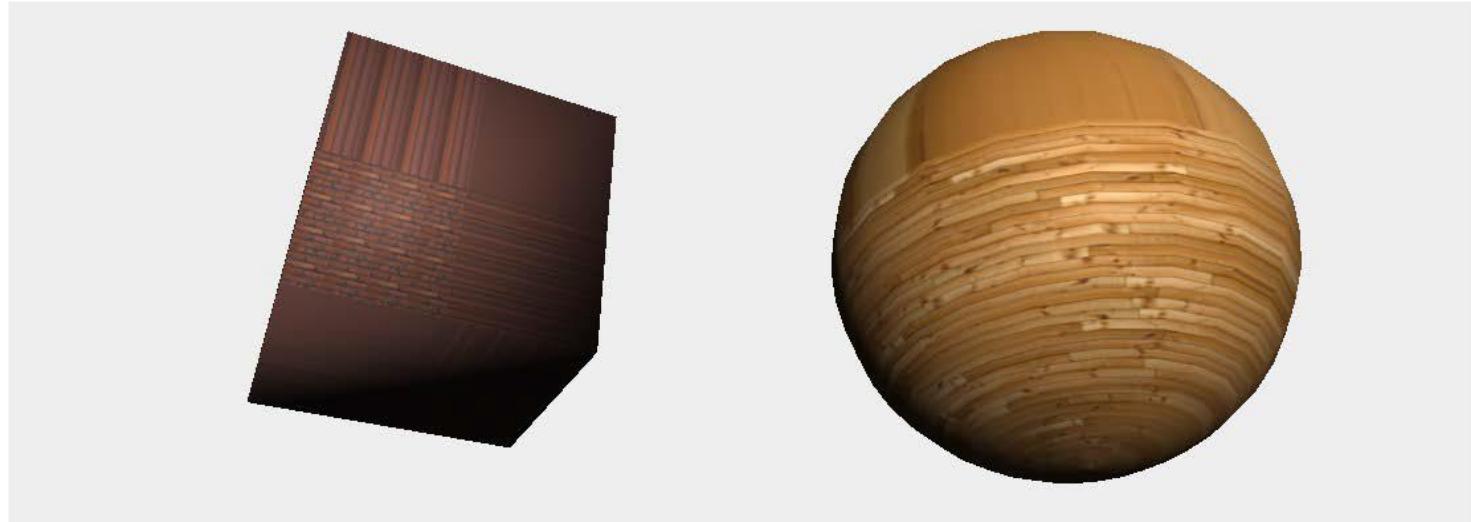
Repeat Wrapping

```
cube.material.map.repeat.set(repeatX, repeatY);
```

- repeatX : how the texture is repeated along XX
- repeatY : the same for the YY axis

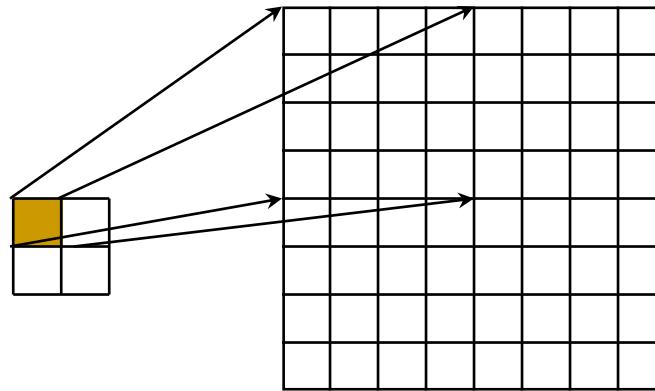
- If **set to 1**, the texture won't repeat itself
- If set to a **higher value**, the texture will start repeating
- Values **less than 1** : zoom in on the texture
- A **negative value** : the texture will be mirrored

Clamping

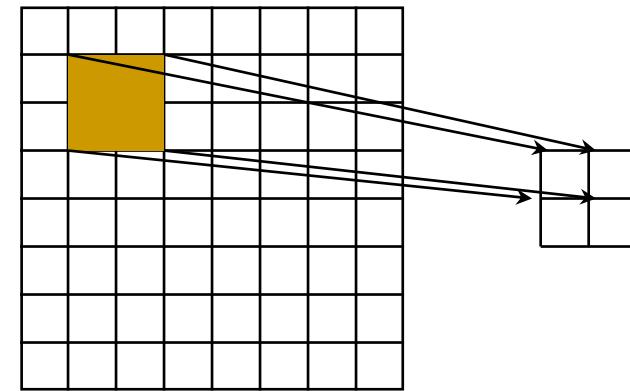


Magnification and Minification

- *Magnification* : more than one pixel can cover a **texel**
- *Minification* : more than one **texel** can cover a pixel
- Can use **point sampling** (nearest texel) or **linear filtering** (2×2 filter) to obtain texture values



Texture
Magnification



Texture
Minification

[Ed Angel]

Magnification and Minification

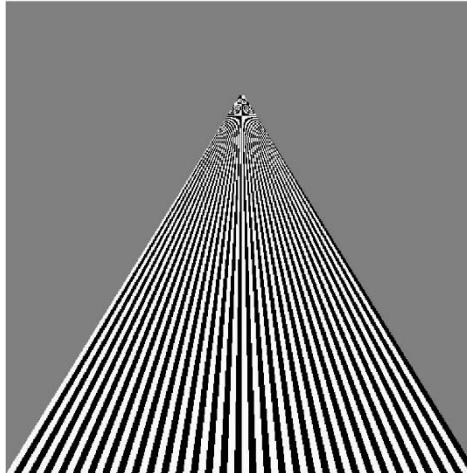
Name	Description
THREE.NearestFilter	This filter uses the color of the nearest texel that it can find. When used for magnification, this will result in blockiness, and when used for minification, the result will lose much detail.
THREE.LinearFilter	This filter is more advanced and uses the color value of the four neighboring texels to determine the correct color. You'll still lose much detail in minification, but the magnification will be much smoother and less blocky.

Mipmapped Textures

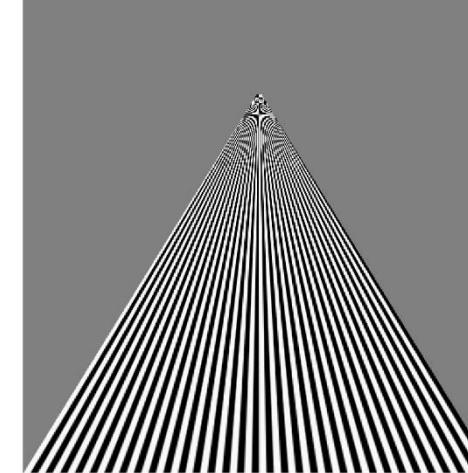
- *Mipmapping* allows for prefiltered texture maps of decreasing resolutions
- Set of texture images, each half the size of the previous one
 - Created when the texture is loaded
 - Allows for smoother filtering
- Lessens interpolation errors for smaller textured objects

Example

point sampling

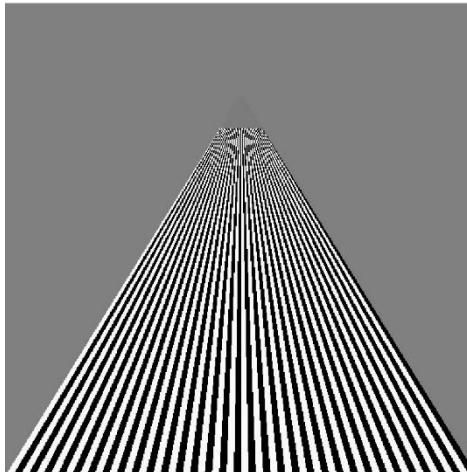


linear filtering

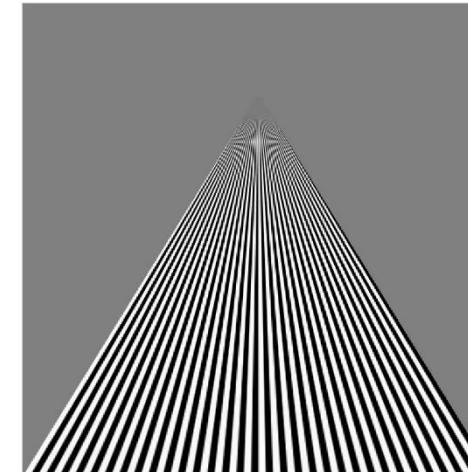


[Ed Angel]

mipmapped
point
sampling



mipmapped
linear
filtering



Mipmapped Textures

Name	Description
THREE.NearestMipMapNearestFilter	This property selects the mipmap that best maps the required resolution and applies the nearest filter principle that we discussed in the previous table. Magnification is still blocky, but minification looks much better.
THREE.NearestMipMapLinearFilter	This property selects not just a single mipmap but the two nearest mipmap levels. On both these levels, a nearest filter is applied to get two intermediate results. These two results are passed through a linear filter to get the final result.
THREE.LinearMipMapNearestFilter	This property selects the mipmap that best maps the required resolution and applies the linear filter principle we discussed in the previous table.

Mipmapped Textures

Name	Description
<code>THREE.LinearMipMapLinearFilter</code>	This property selects not a single mipmap but the two nearest mipmap levels. On both these levels, a linear filter is applied to get two intermediate results. These two results are passed through a linear filter to get the final result.

What can we do ?

- Use a **bump map** to create bumps / wrinkles

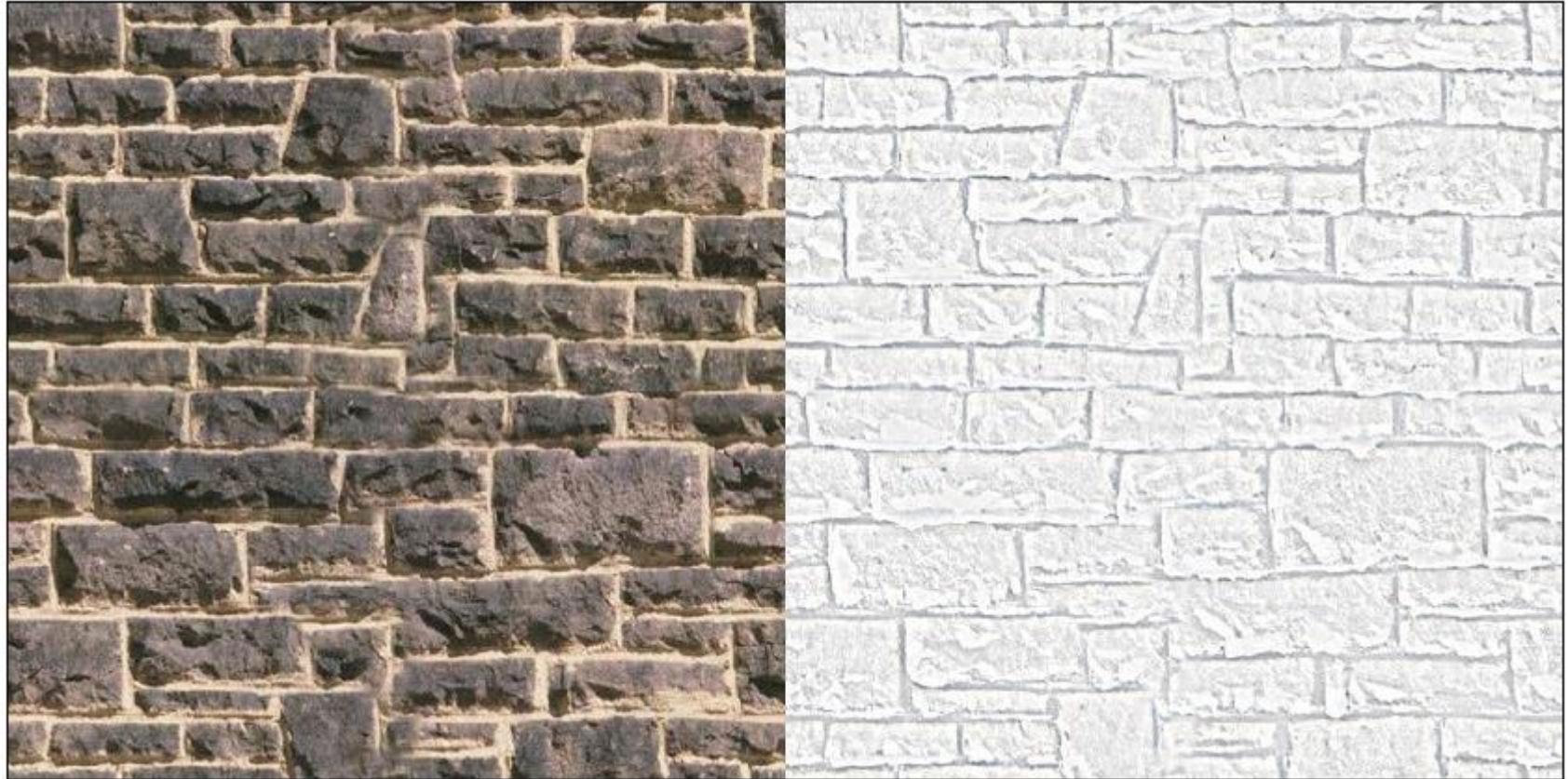


[Dirksen]

Loading and applying a bump map

```
function createMesh(geom, imageFile, bump) {  
    var texture = THREE.ImageUtils.loadTexture(imageFile);  
    var mat = new THREE.MeshPhongMaterial();  
    mat.map = texture;  
    if (bump) {  
        var bump = THREE.ImageUtils.loadTexture(bump);  
        mat.bumpMap = bump;  
        mat.bumpScale = 0.2;  
    }  
    var mesh = new THREE.Mesh(geom, mat);  
    return mesh;  
}
```

Texture and Bump Map

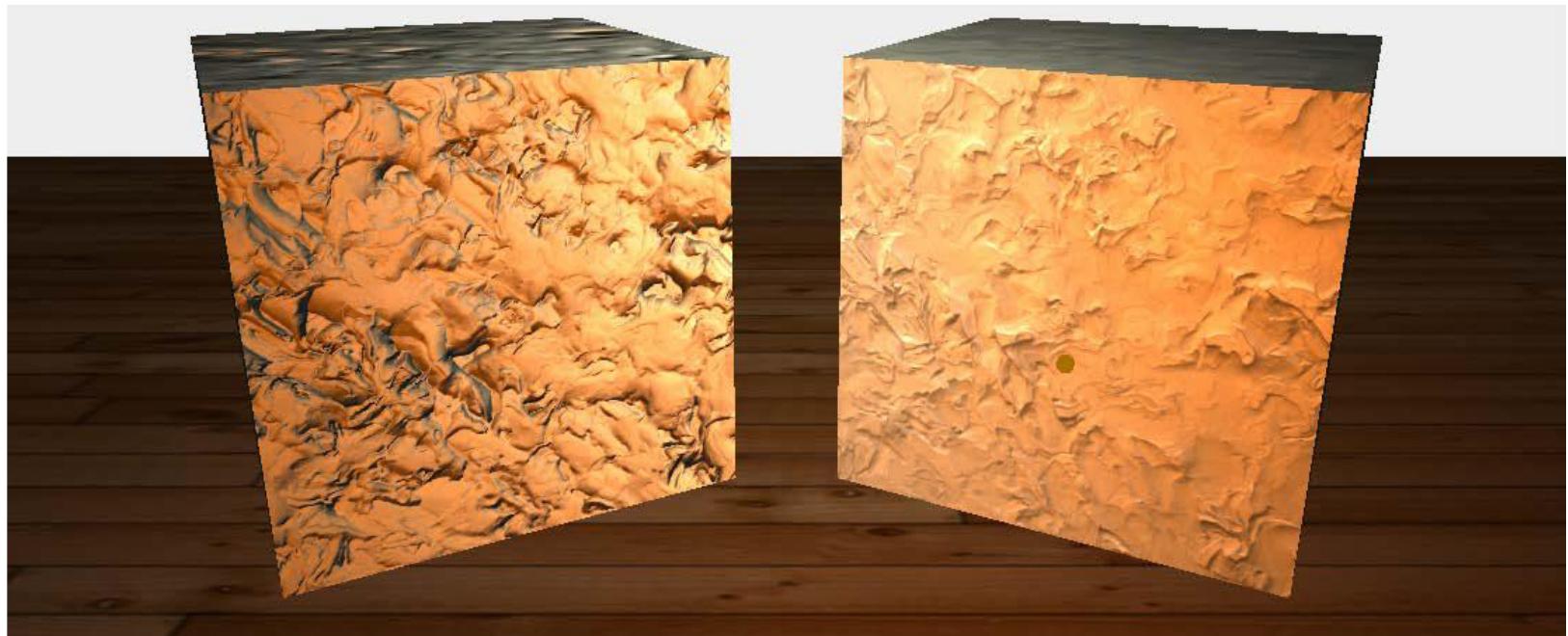


The Bump Map

- Grayscale or color image
- Intensity of each pixel defines the height of the bump
- Only contains the relative height of a pixel
 - No direction / slope information
- Limited level of detail and depth perception
- For finer detail, use a normal map

What can we do ?

- Use a **normal map** to achieve more detail

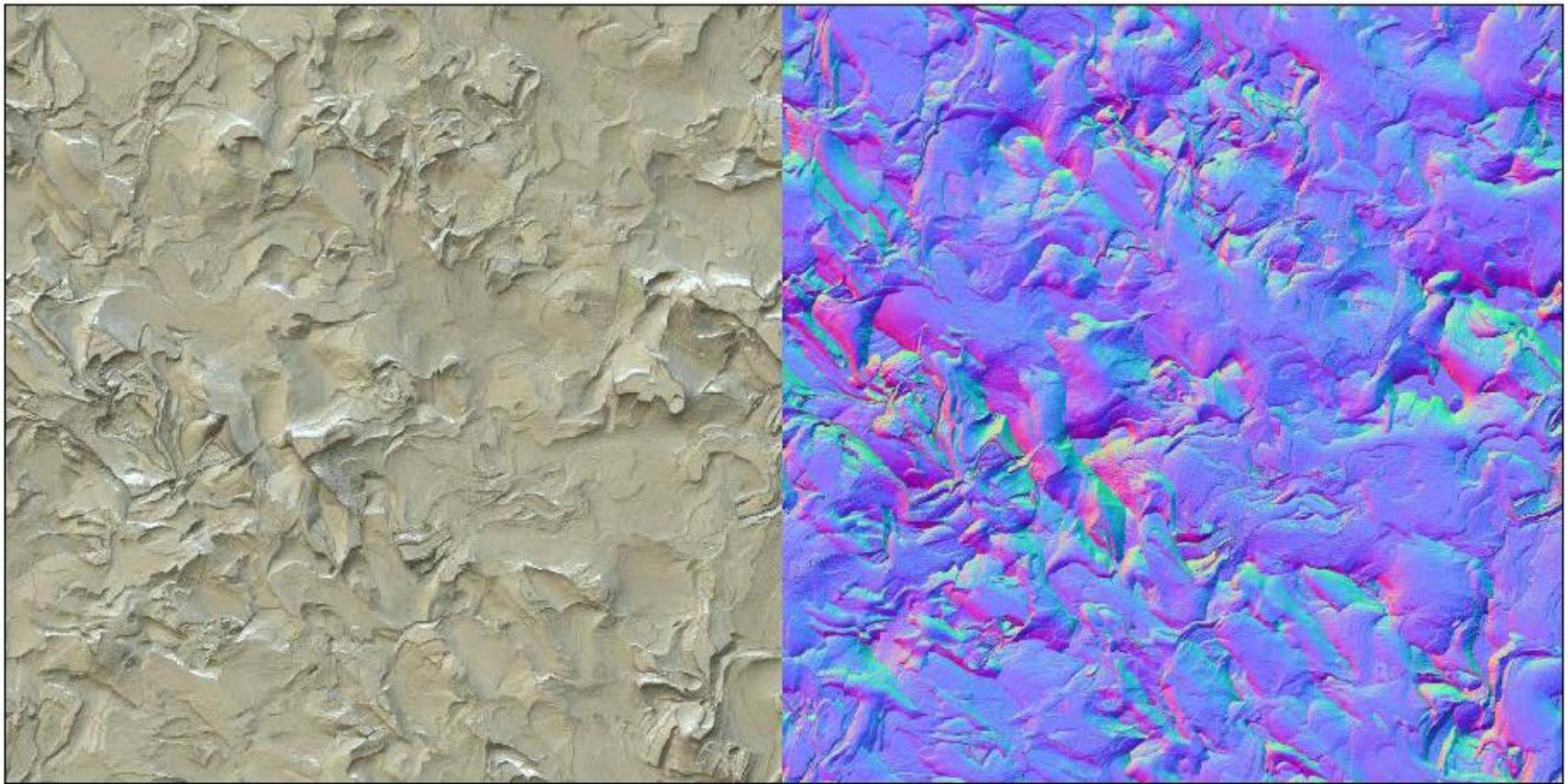


[Dirksen]

Loading and applying a normal map

```
function createMesh(geom, imageFile, normal) {  
    var t = THREE.ImageUtils.loadTexture(imageFile);  
    var m = THREE.ImageUtils.loadTexture(normal);  
  
    var mat = new THREE.MeshPhongMaterial();  
    mat.map = t;  
    mat.normalMap = m;  
  
    var mesh = new THREE.Mesh(geom, mat);  
    return mesh;  
}
```

Texture and Normal Map

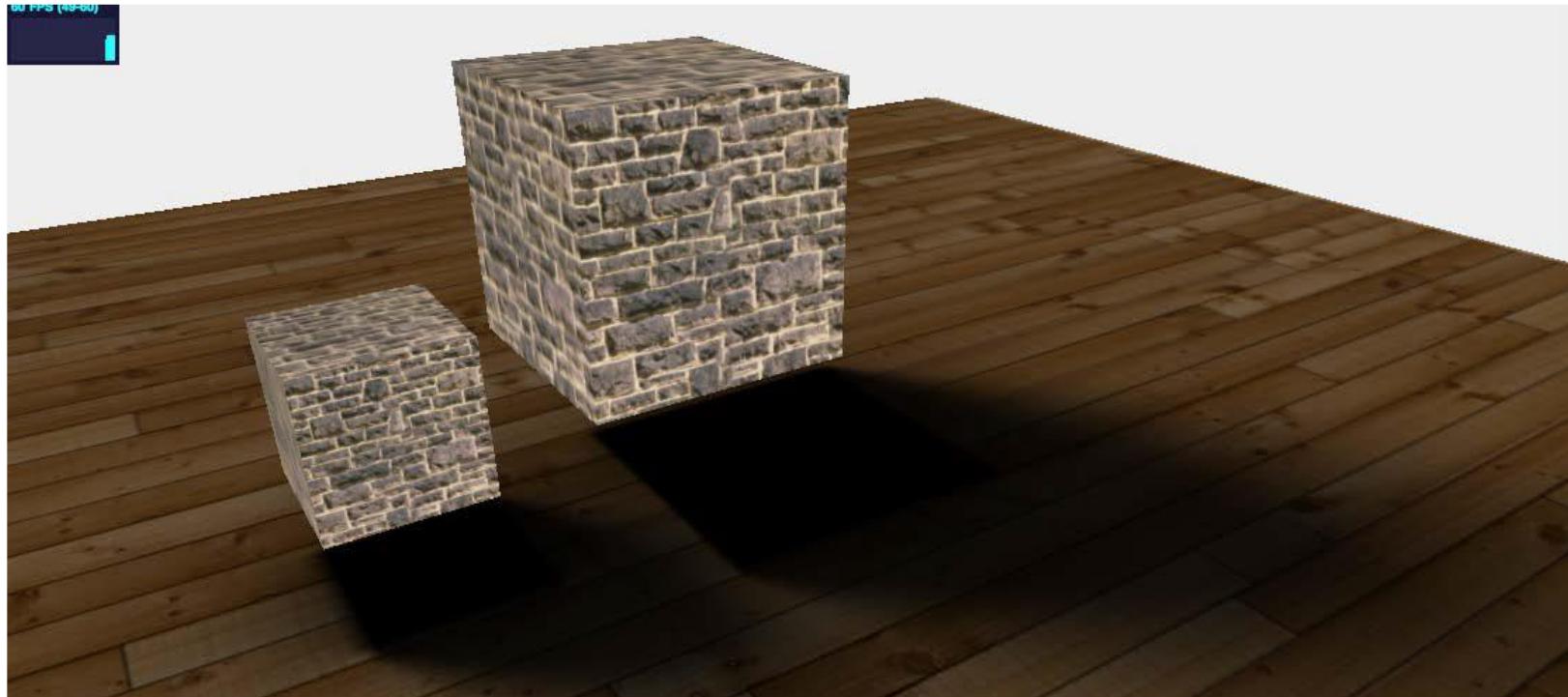


The Normal Map

- Not easy to create !
- Use specialized tools, such as Blender or Photoshop
 - Create normal maps from high-resolution renderings or textures
- Three.js provides a way to do this during runtime
 - `getNormalMap()` in `THREE.ImageUtils`
 - Take a JavaScript/DOM image as input and convert it into a normal map

What can we do ?

- Create fake **shadows** using a **light map**



[Dirksen]

What can we do ?

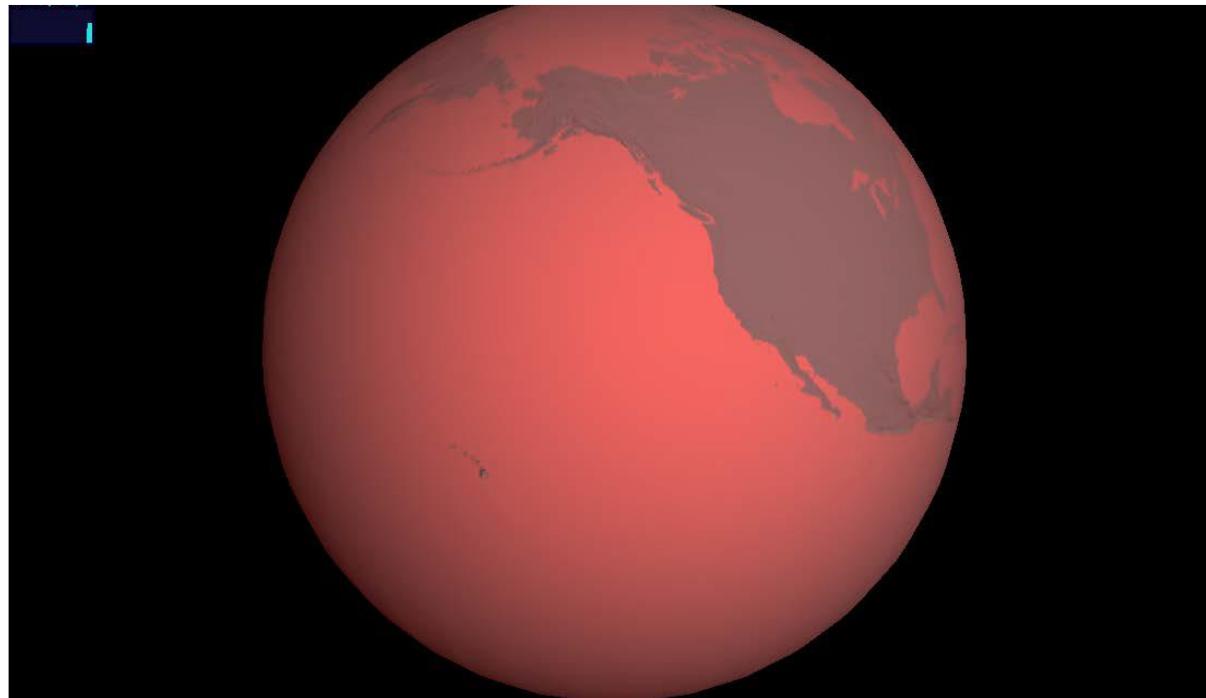
- Create fake **reflections** using an **environment map**



[Dirksen]

What can we do ?

- Specify the **shininess** and the **highlight color** of a material with a **specular map**



[Dirksen]

threejs.org – Examples

- Check the examples !!
- <https://threejs.org/examples/?q=texture>

Acknowledgments

- Some ideas and figures have been taken from slides of other CG courses.
- In particular, from the slides made available by Ed Angel and Andy van Dam.
- As well as from the Three.js book by Jos Dirksen
- Thanks !