Interactive Computer Graphics

CS GY-6533

Prof. Ivan Safrin

Housekeeping

Lectures:	Tues. 3:00-5:30pm
Classroom:	2 MTC 08.812 (8th Floor)
Software Eng. Lab:	RH223
Course Web Site:	`https://github.com/ivansafrin/CS6533`

Instructor:	Prof. Ivan Safrin
Office:	2 MTC, 08.812 (8th Floor)
Office Hours:	Tues. 5:30-7:00pm
E-mail:	`is1296@nyu.edu`

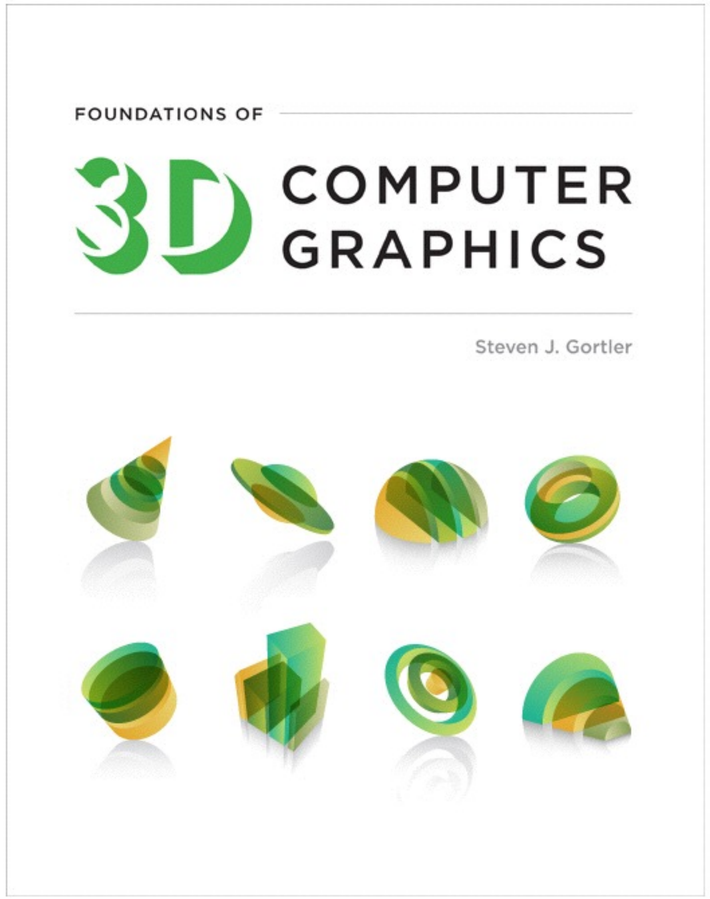
TA: TBA.

Objectives: This course aims to provide a broad introduction to the field of Computer Graphics, and to describe the techniques that are commonly used in the graphics industry today (such as in production of special effects, computer animation, video games, and virtual reality). This course is combination of algorithms, numerical methods, representations and models of the shape and appearance of real-world objects, and methods for their display and manipulation. It involves a lot of programming, and requires a certain degree of mathematical sophistication (in linear algebra, specifically). But it's also a lot of fun. No artistic skill is required, but it does come in handy.

Prerequisites: CS2134 (Data Structures and Algorithms) or equivalents, knowledge of C++ programming, and a familiarity with Linear Algebra and Calculus.

Expected Work: Students will design, implement, and use interactive graphical applications (using the OpenGL API in C++). This amounts to four coding assignments using the OpenGL C++ API and the OpenGL Shading Language (GLSL). Some of the assignments build off of each other.

Exams: There will be one midterm and a final exam. The exams will be based on the material covered in class, and on what is learned from completing the assignments.



Textbook

Foundations of 3D Computer Graphics

Steven Gortler

MIT Press

Grading: Midterm: 25%, Final: 25%, 4 Programming Assignments: 50%. Highest grade (A): >= 90% of the points. Fail (F): < 65% of the points. Grade changes will only be issued in cases of calculating and recording errors. Once a final grade has been submitted, it is considered final, see `http://bulletin.engineering.nyu.edu/`.

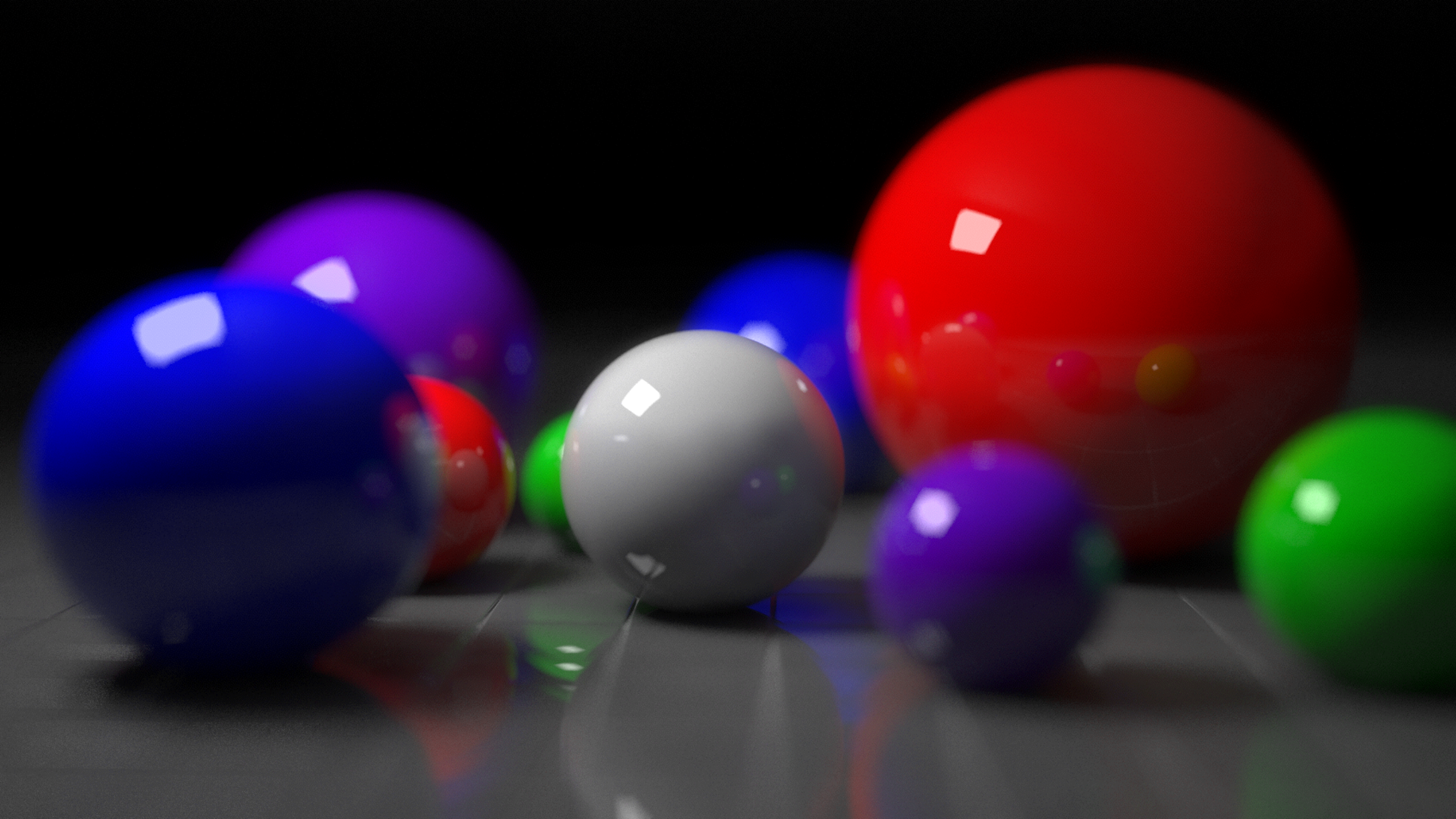
General Instructions for Programming Assignments: Submit your write-up, your source code (with full comments and documentation), and include your e-mail address as well as brief instructions on how to compile and run your programs. You can submit your assignments via email or via a github repository. Important: You MUST use the skeleton code provided NOTE: You may discuss the programming assignments with other students currently taking the course, BUT EACH WRITE-UP AND PROGRAM MUST BE DONE INDIVIDUALLY AND INDEPENDENTLY, AND YOU SHOULD SHOW THAT YOU PERSONALLY UNDERSTAND EVERYTHING THAT YOU SUBMIT. IN OTHER WORDS, CODE SIMILARITIES WILL BE CONSIDERED CHEATING, AND THE ASSIGNMENT WILL BE GRADED WITH ZERO POINTS. The incident will furthermore be reported to the dean of student affairs, possibly resulting in an F grade for the class, see

`http://engineering.nyu.edu/academics/code-of-conduct/academic-dishonesty`

Late Policy: For late projects and homework, 5% of the grade will be deducted for each day the project is not submitted.

Surprise Quiz

Computer Graphics



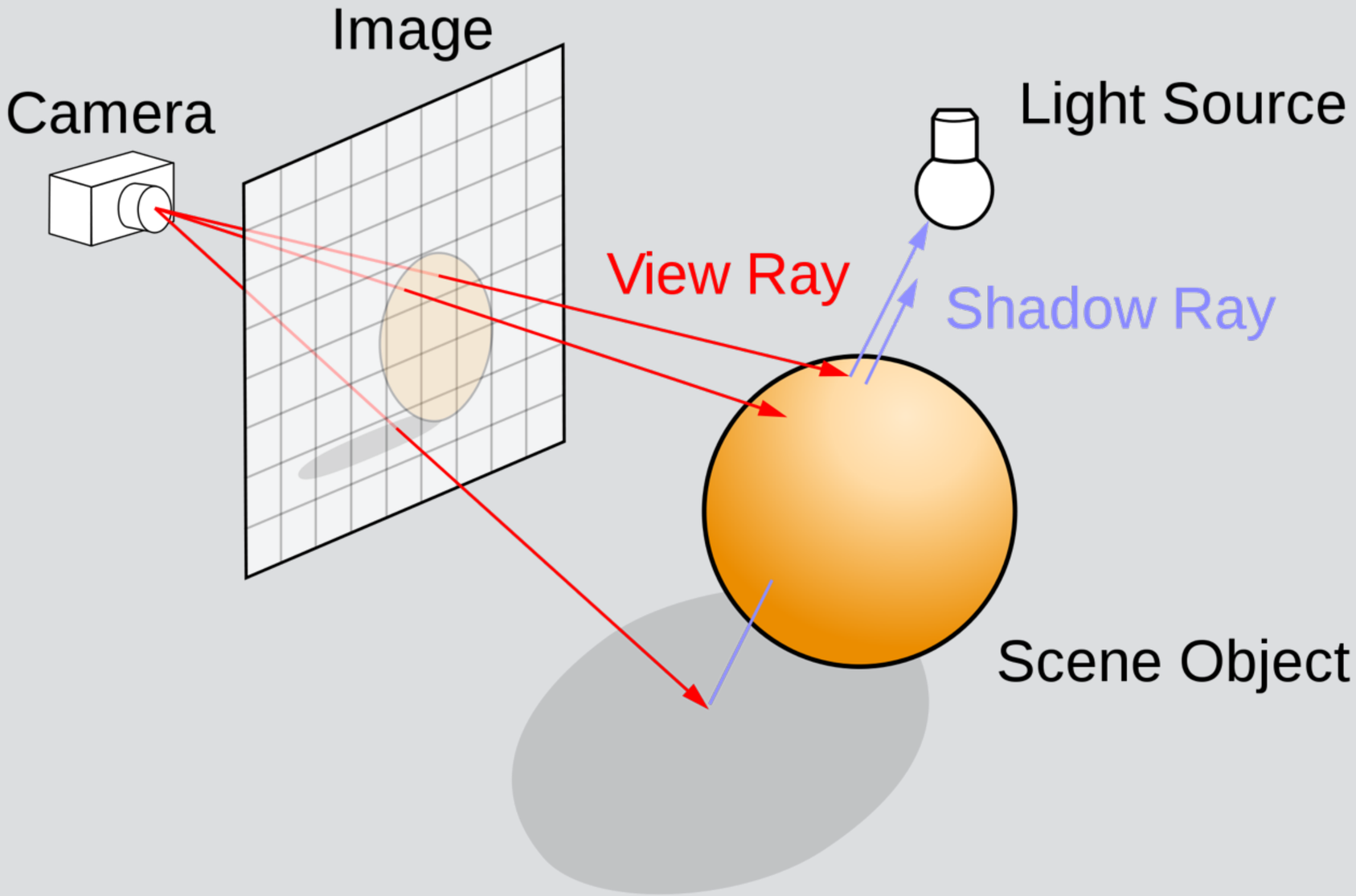
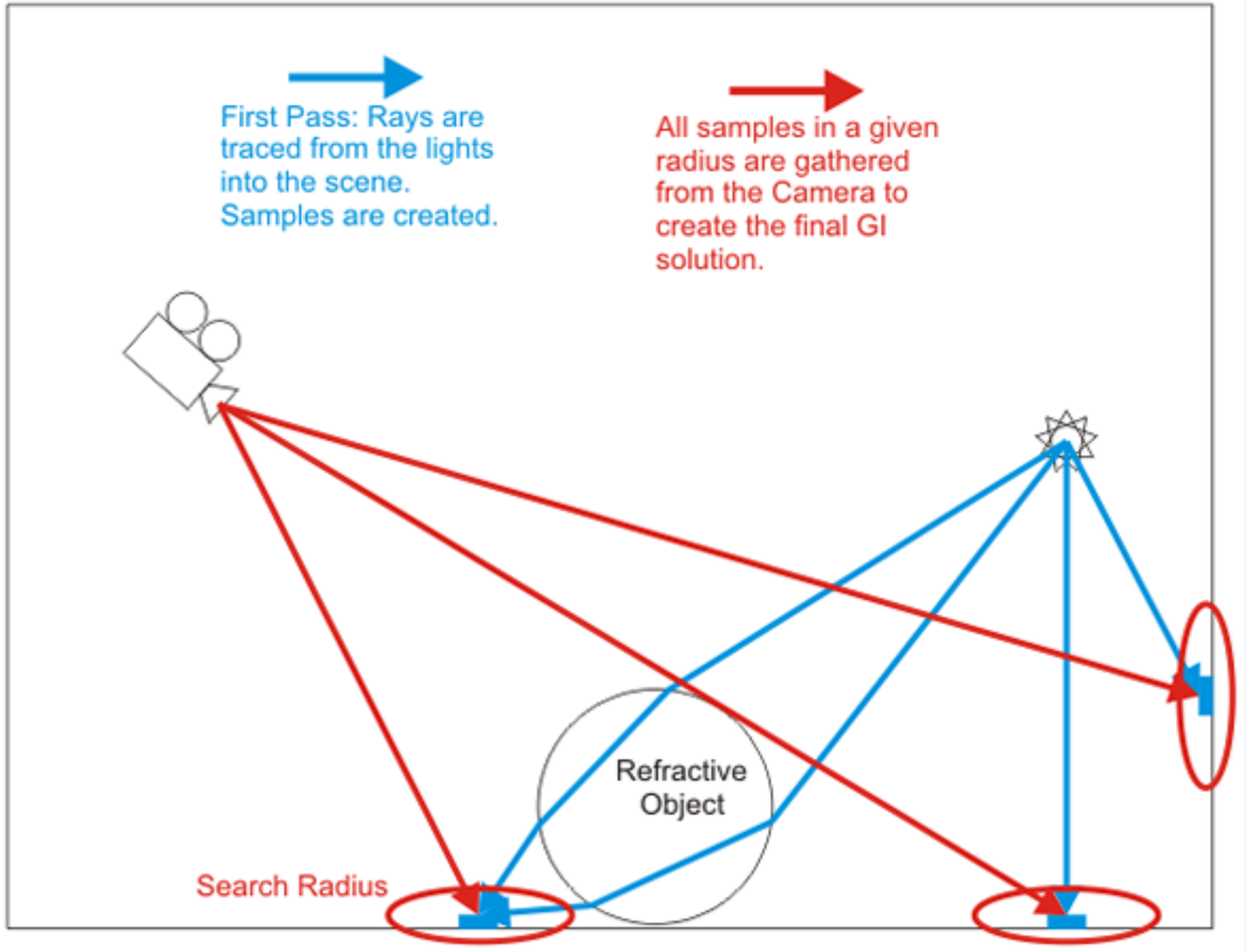


Image
Camera
Light Source
View Ray
Shadow Ray
Scene Object


First Pass: Rays are traced from the lights into the scene. Samples are created.
All samples in a given radius are gathered from the Camera to create the final GI solution.
Refractive Object
Search Radius

The use of interactive computer graphics.

Appearance

Appearance Presets

manix_small

Density Scale	100.000
Shading Type	BRDF Only
Gradient Factor	3.000
Primary Step Size	3.00
Secondary Step Size	3.00

Transfer Function

diffuse
specular
emission

Node Selection

Node 4

Node Properties

Intensity	102.97
Opacity	1.000
Diffuse	
Specular	
Glossiness	0.000
Emission	

Camera

Camera Presets

manix_small

Film

Type	HDTV (video)		
Film width	1920	1920 x 1080	480 x 270
Film height	1080	1280 x 720	320 x 180
Exposure	0.75		

Noise Reduction

Aperture

Size 0.000 mm

Projection

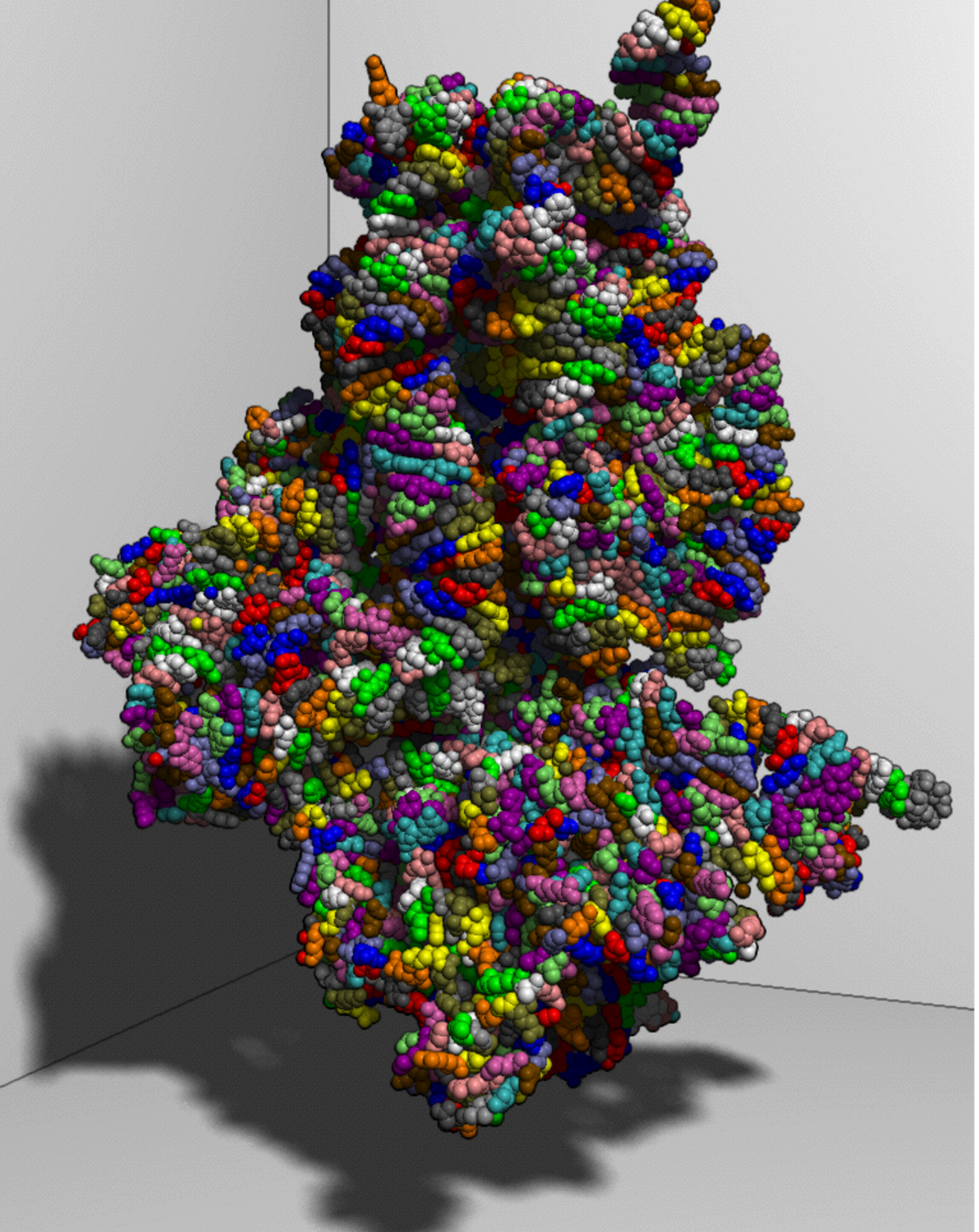
Field of view 35.00 deg.

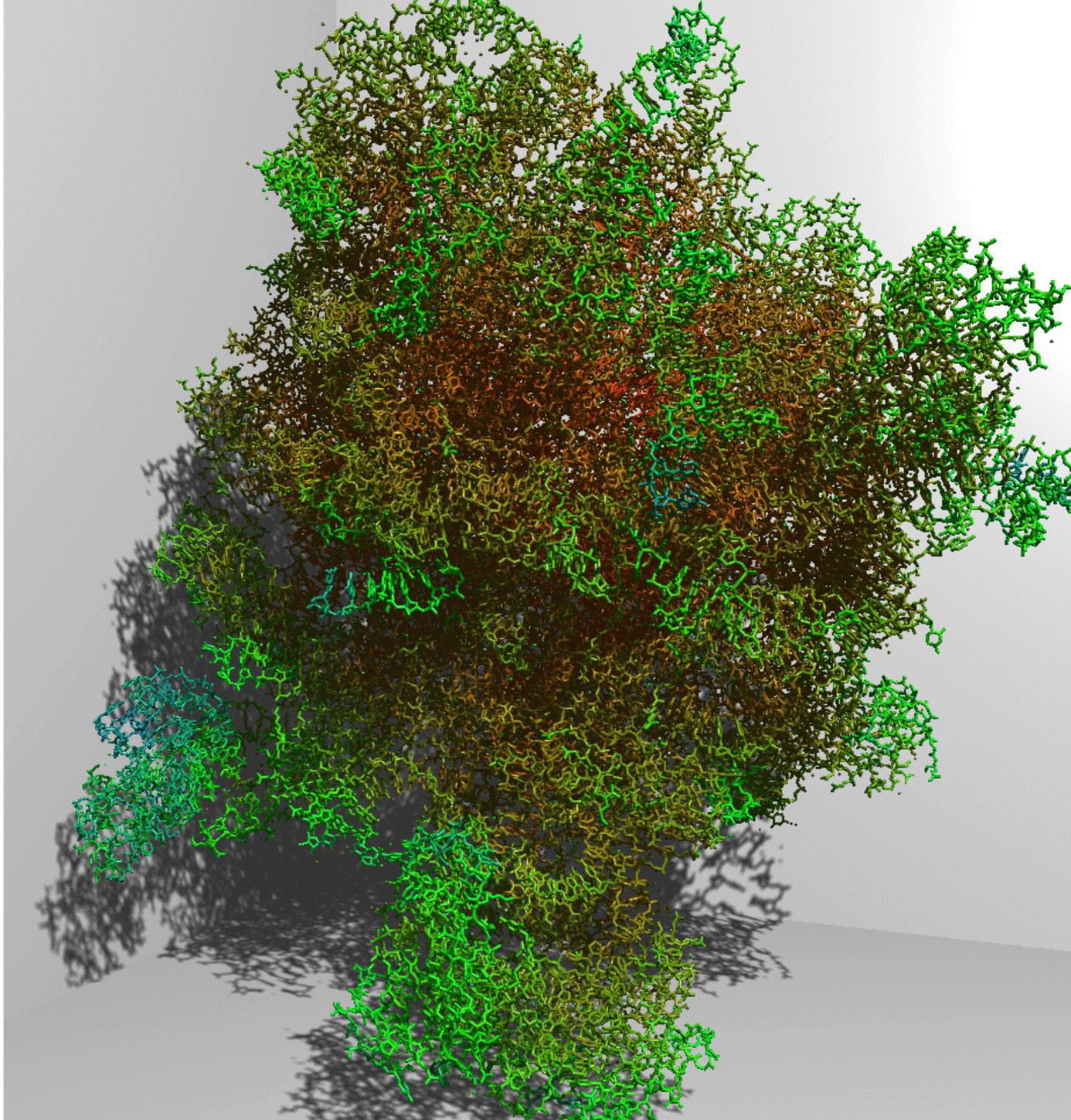
Log

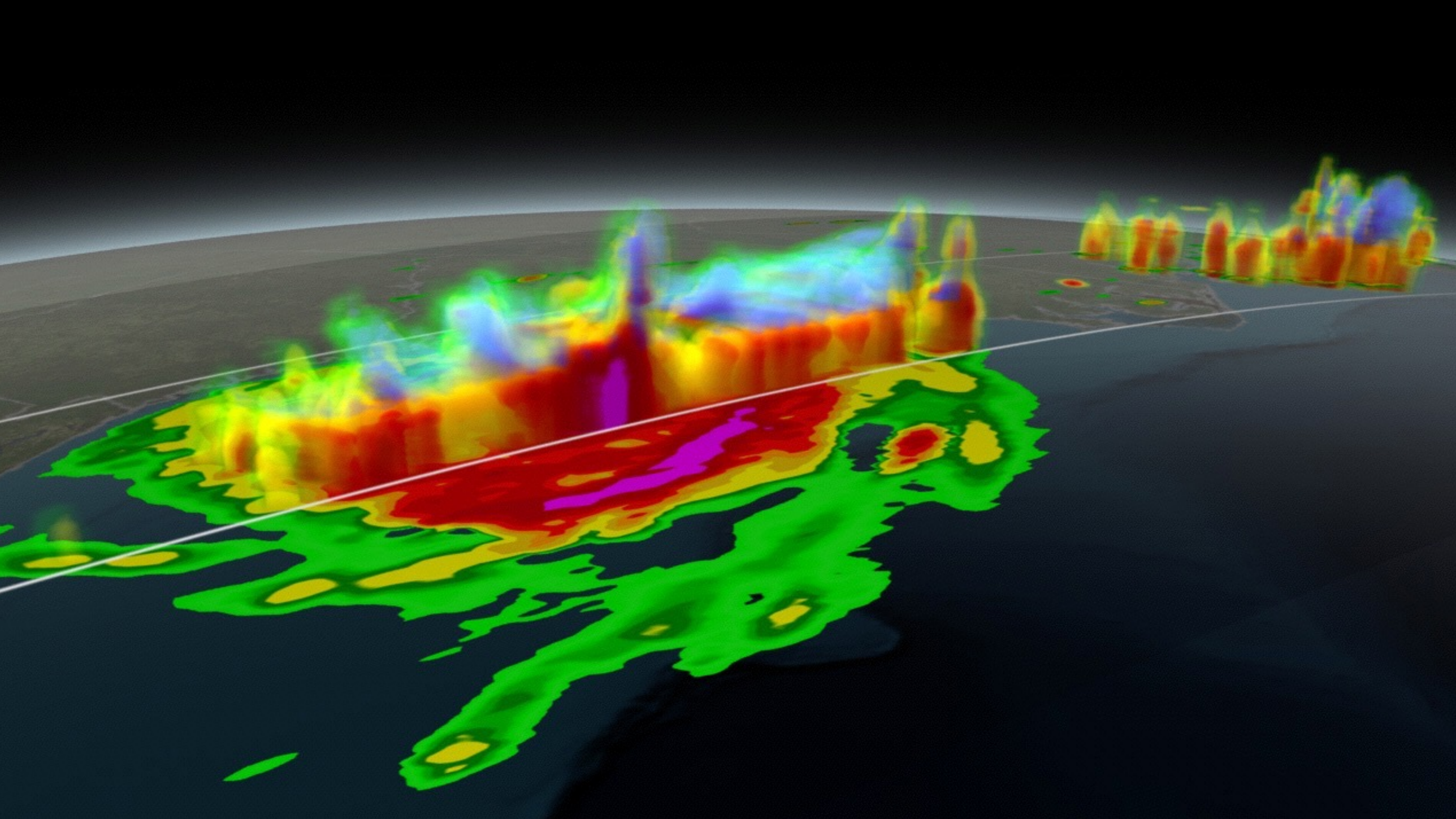
16:47:43	Render canvas resized to: 1920 x 1080 pixels
16:47:37	Render canvas resized to: 400 x 500 pixels
16:47:37	Render canvas resized to: 640 x 480 pixels
16:47:37	Device memory: 722.75/-64.00 MB
16:47:37	Rendering started
16:47:37	Copying gradient magnitude to device
16:47:37	Copying density volume to device
16:47:37	Creating histogram
16:47:37	Creating density histogram
16:47:37	Mean gradient magnitude: 35.95
16:47:36	Creating gradient magnitude histogram
16:47:36	Gradient magnitude range: [0.00 - 3017.00]

Statistics

Performance	
Timings	
Render Image	59.99
Blur Estimate	3.81
Post Process Estimate	1.82
De-noise Image	4.54
FPS	13.00
No. Iterations	76
Volume	
File	manix_small.mhd
Bounding Box	
Min	[0.00, 0.00, 0.00]
Max	[0.78, 1.00, 0.78]
Physical Size	250.00 x 322.00 x 250.00
Resolution	256 x 230 x 256
Spacing	0.98 x 1.40 x 0.98
No. Voxels	15073280
Density Range	[-1000, 3095]
Memory	
CUDA Memory	
Density Buffer	28.75
Gradient Magnitude Buffer	28.75
Scene	0.01
Frame Buffers	
Host Memory	
LDR Frame Buffer	17.80

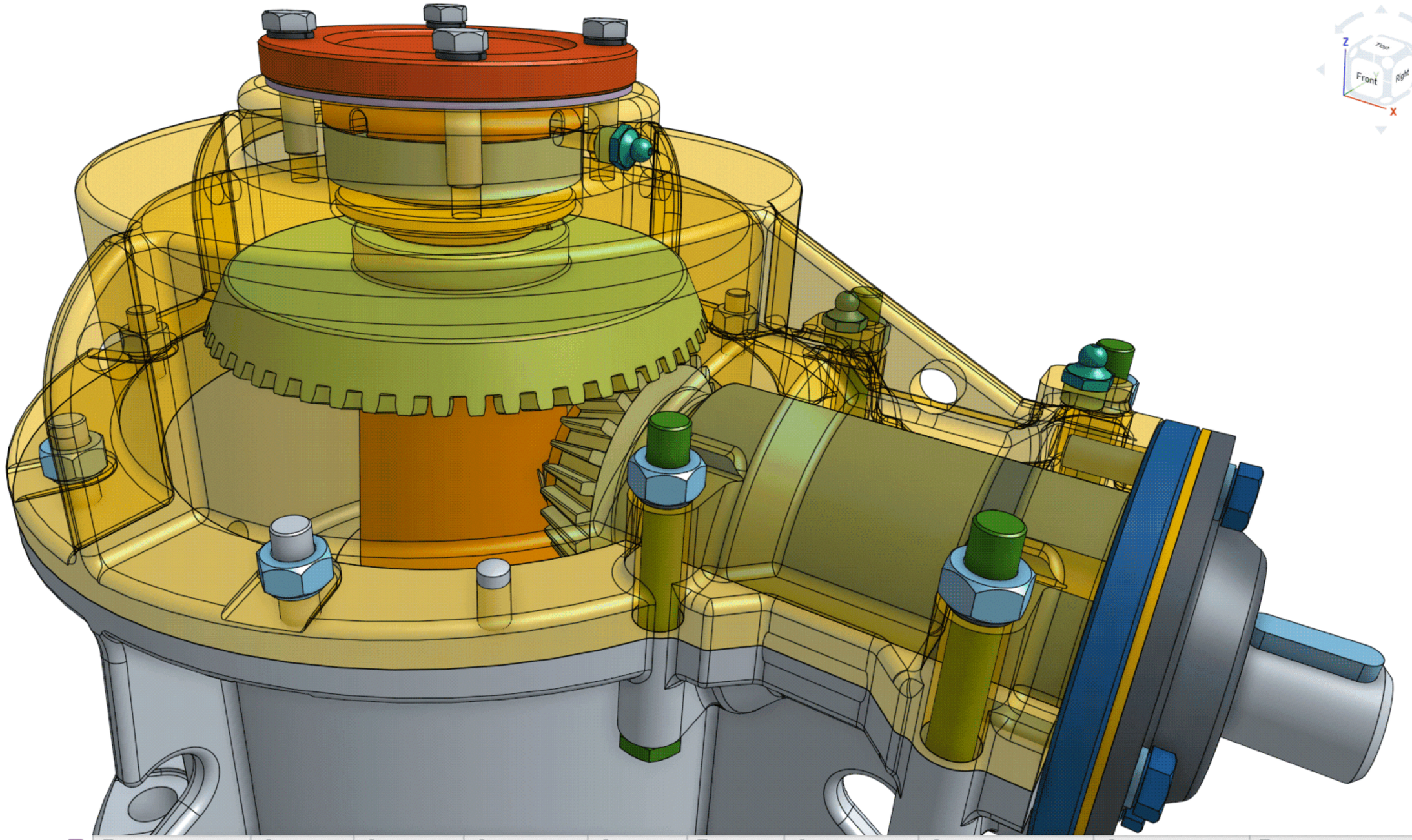




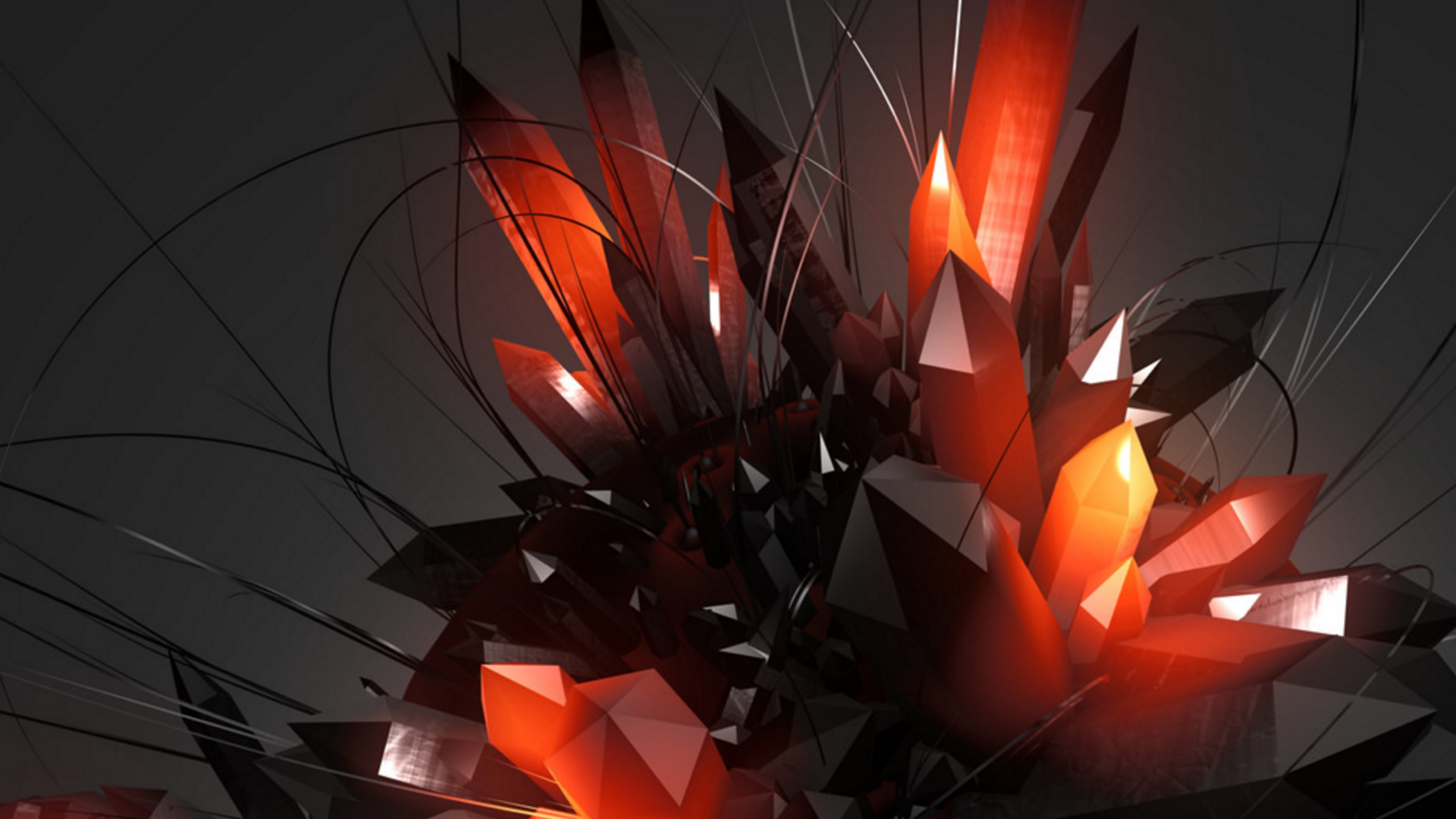


Instances (50)

- Origin
- Bottom case <1>
- Dowel PN-89_M-8502...
- Top case <1>
- Vertical gear assembl...
- Horizontal gear asse...
- Bevel Gear11 <1>
- Bevel Gear21 <1>
- PN-77_M-82008 - 8,2 ...
- PN-77_M-82008 - 8,2 ...
- PN-77_M-82008 - 8,2 ...
- PN-77_M-82008 - 8,2 ...
- ISO 4017 - M8 x 35Dl...
- ISO 4017 - M8 x 35Dl...
- ISO 4017 - M8 x 35Dl...
- ISO 4017 - M8 x 35Dl...
- PN-76_M-86003 - M1...
- PN-76_M-86003 - M1...
- WSKA_NIK_POZ_OLE...
- ISO 4017 - M8 x 35Dl...
- ISO 4017 - M8 x 35Dl...
- ISO 4017 - M8 x 35Dl...

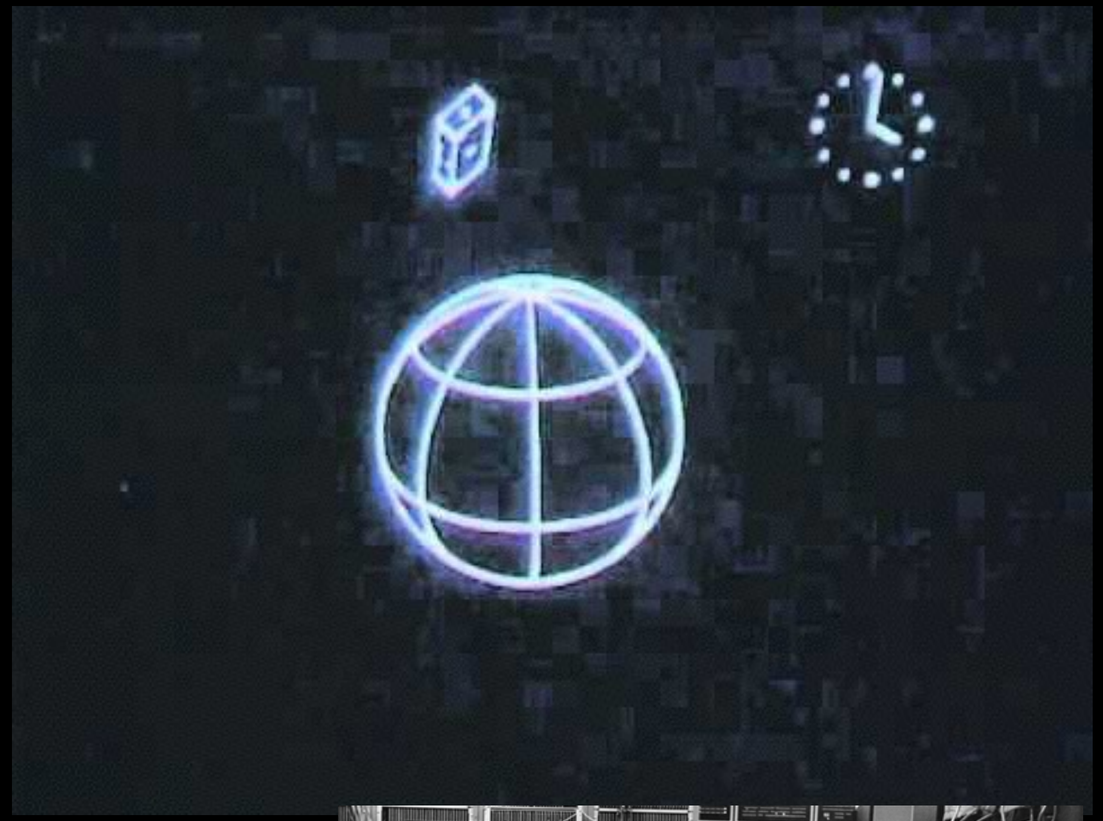


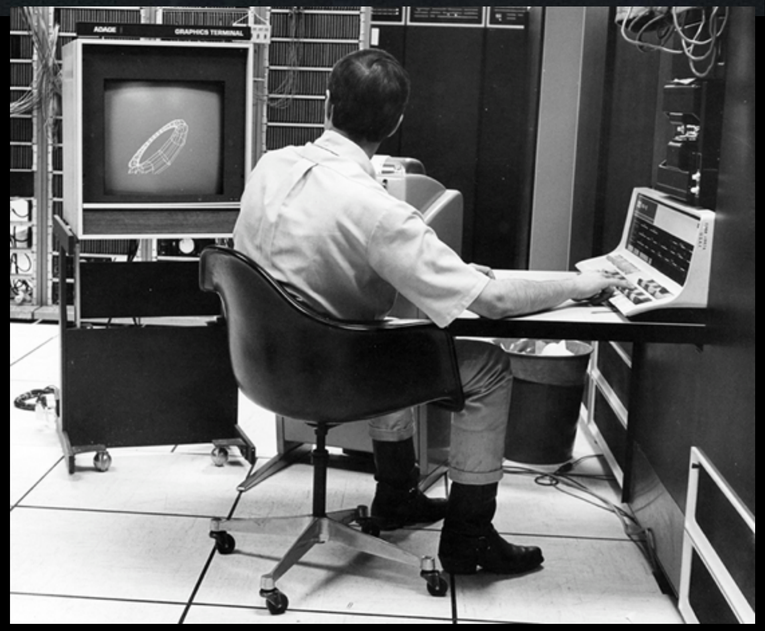

SEAT MUST BE IN
FORWARD 7 INCHES OF TRAVEL
DURING TAKEOFF AND LANDING
SEAT MUST BE IN
FORWARD 7 INCHES OF TRAVEL
DURING TAKEOFF AND LANDING



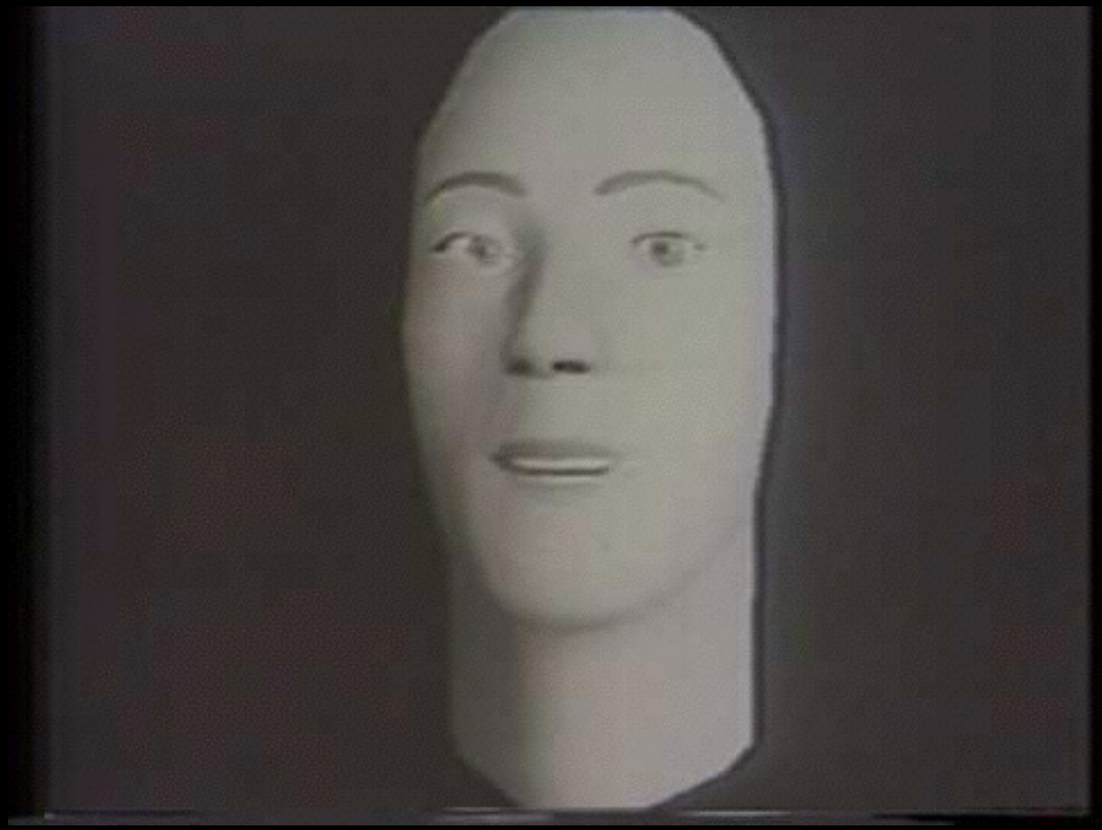

JUN SAN ANDREAS
29IAI791

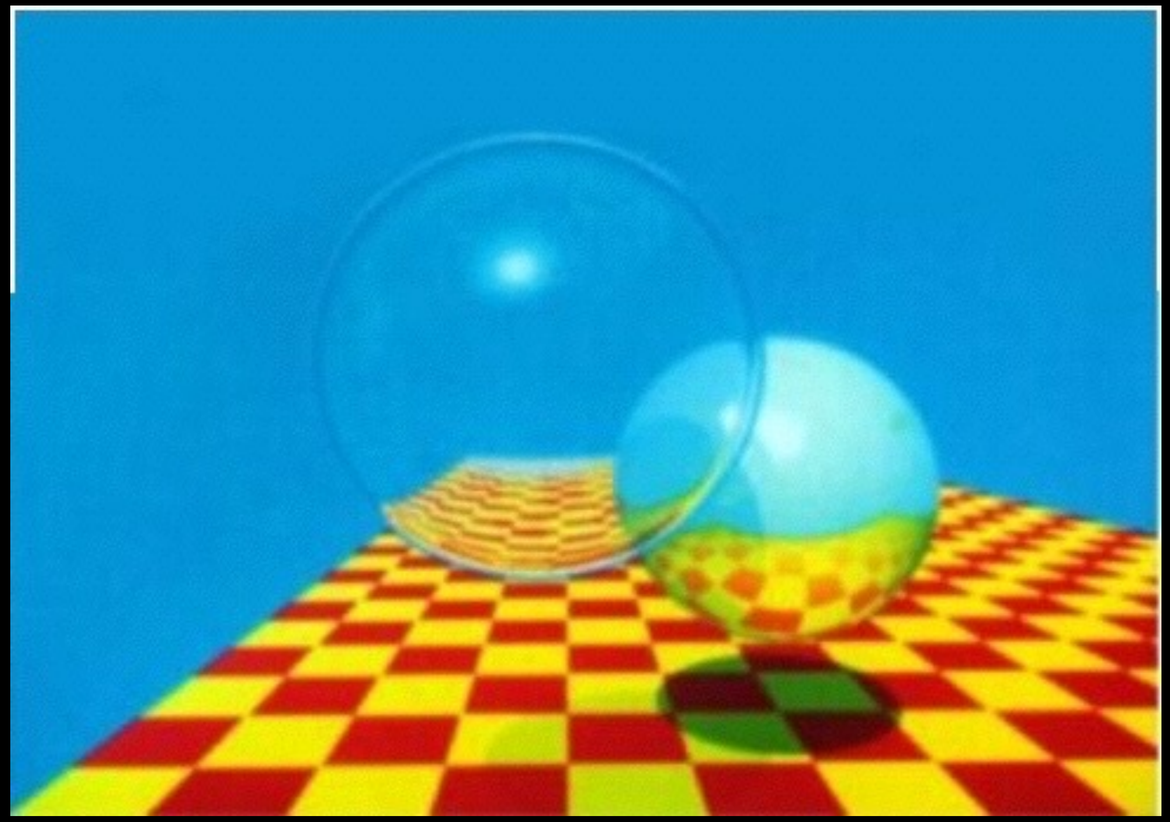
A (very) brief **history** of computer graphics.

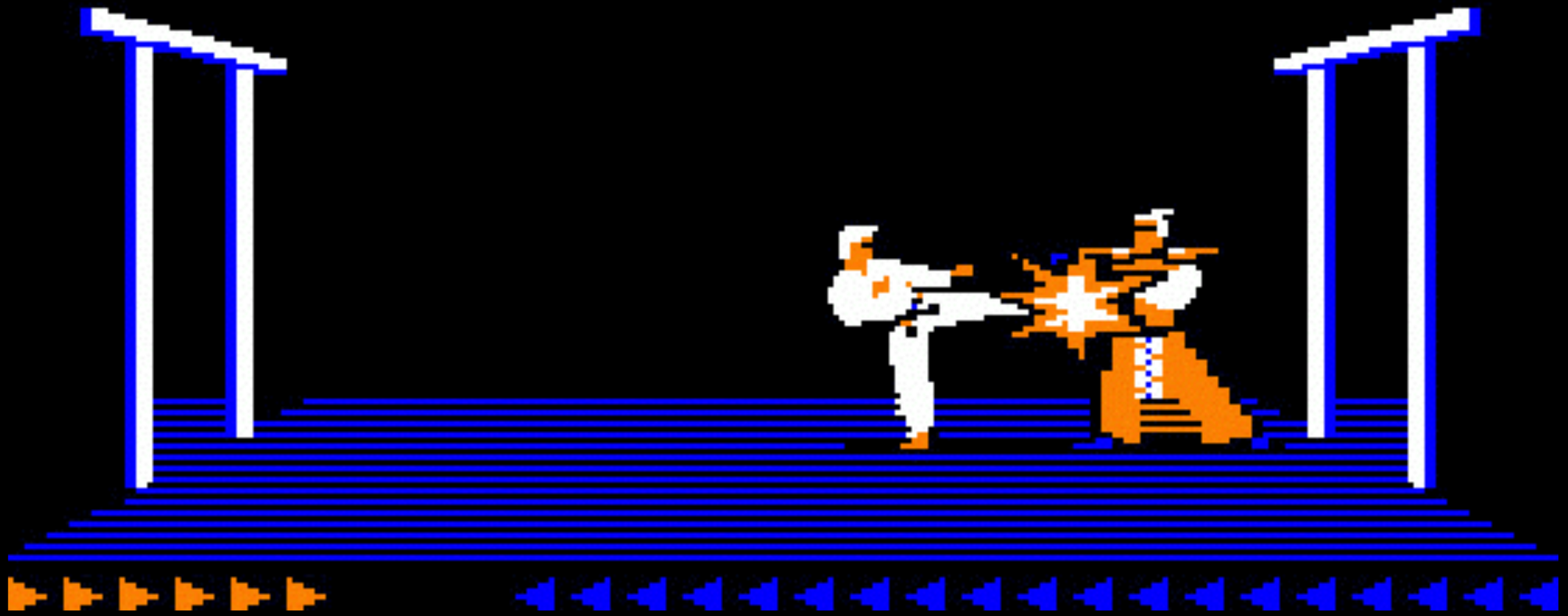


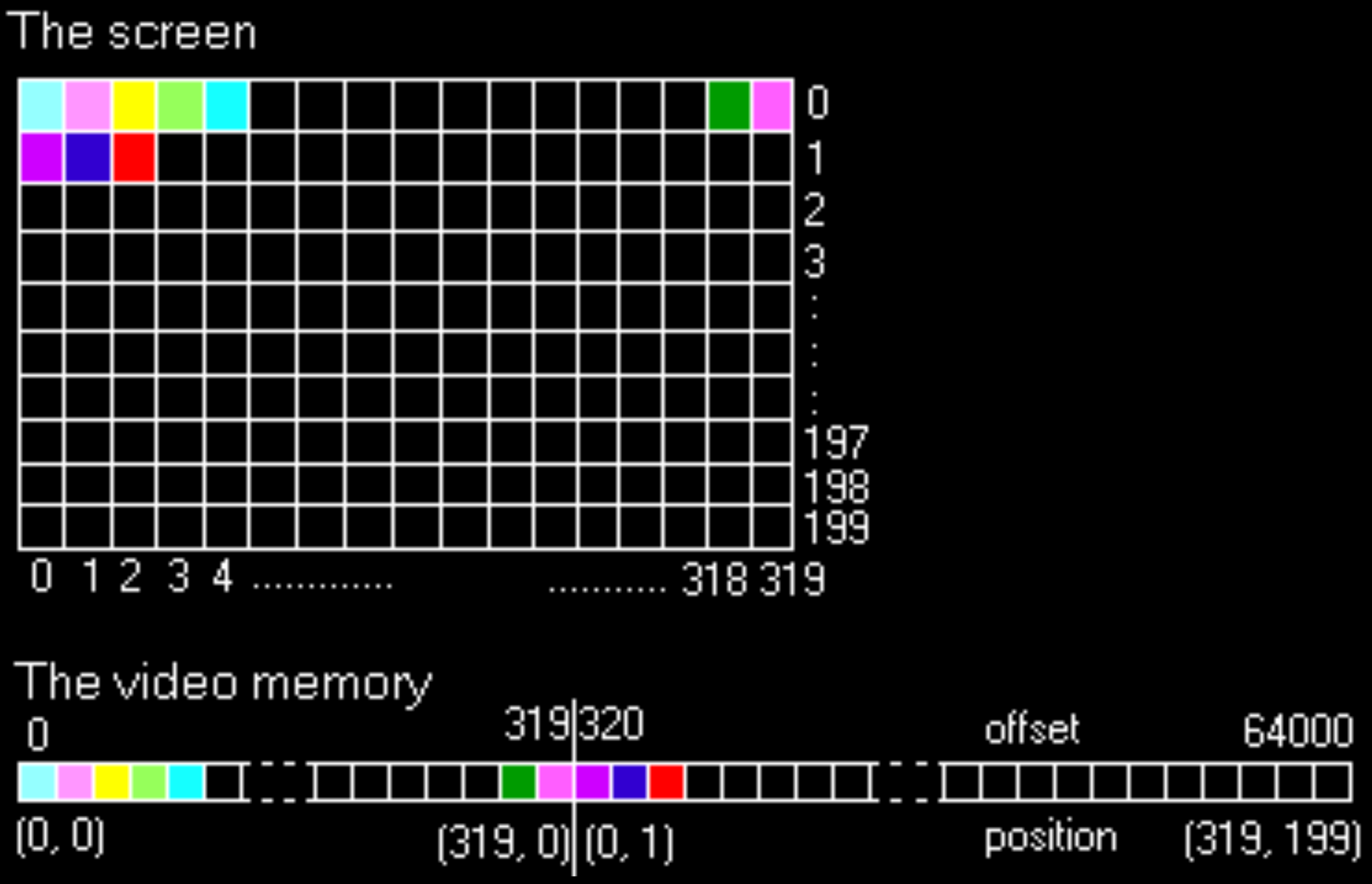

ADAGE
GRAPHICS TERMINAL

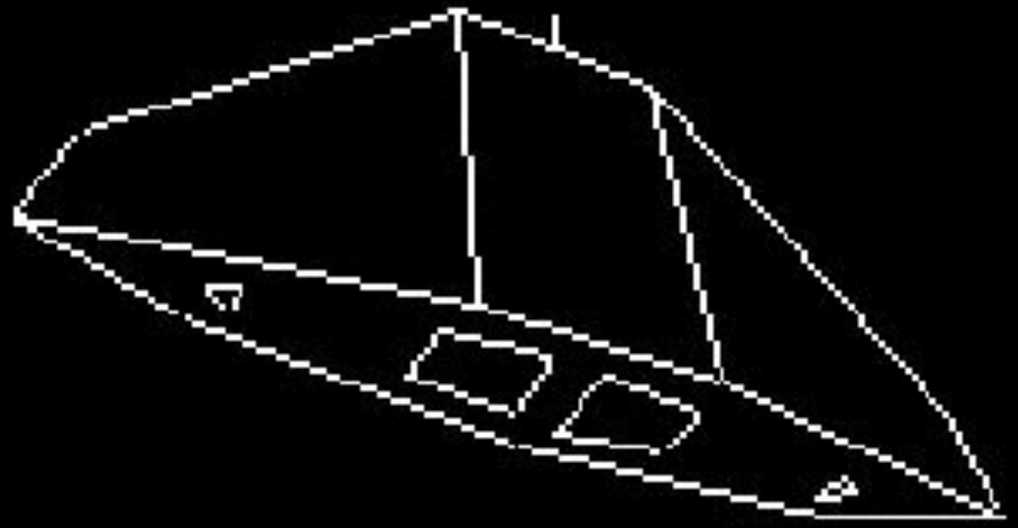

DAC-1
DELETE A REMOVE







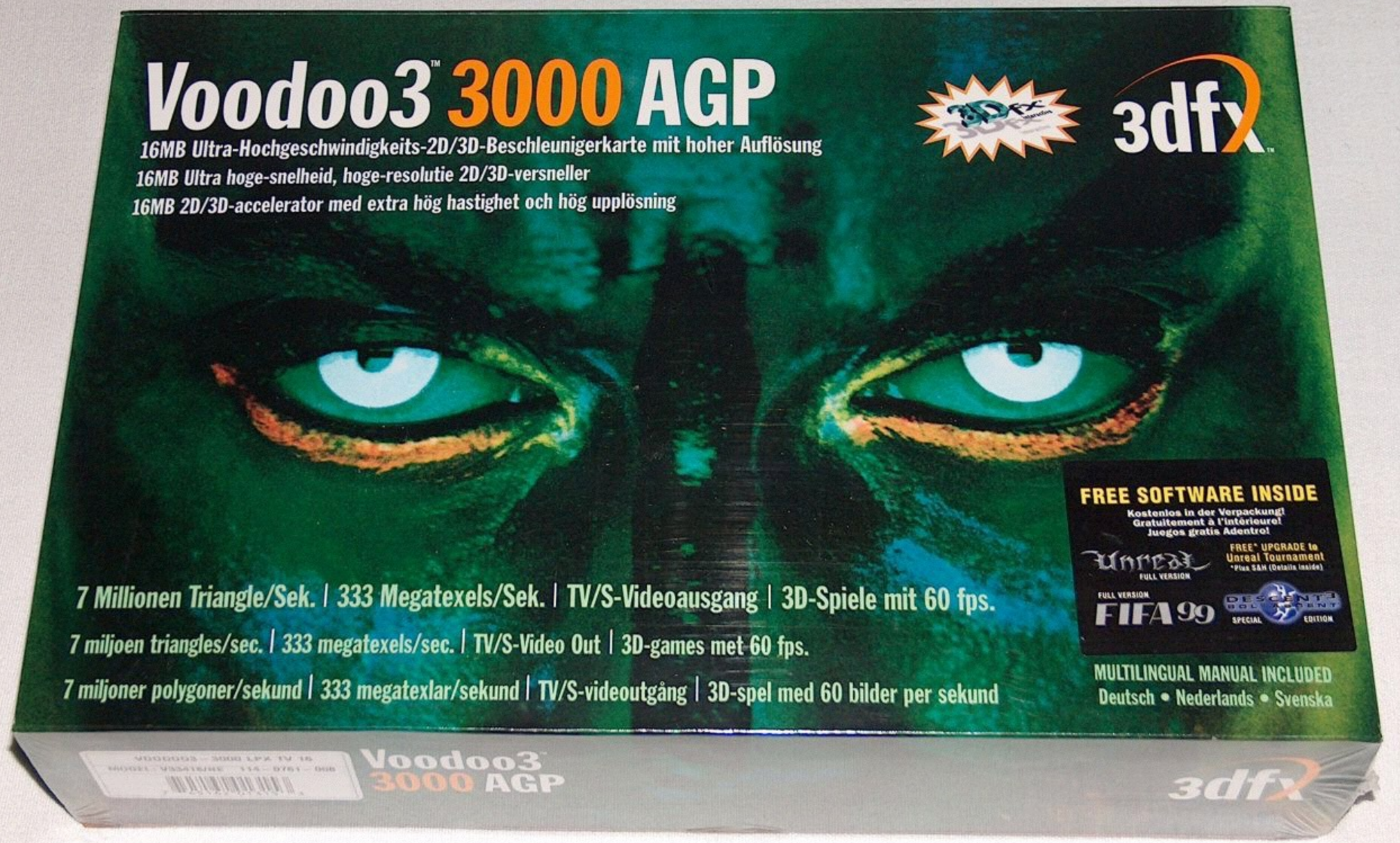

The screen
0
1
2
3
197
198
199
0 1 2 3 4 .... 318 319
The video memory
0
319
320
offset
64000
(0, 0)
(319, 0)
(0, 1)
position
(319, 199)



Load New Commander (Y/N)?


SPD: 120
VSI: 00.59
THR:70%
HDG:251
ALT:401
ZOOM:2


93
75
13
0
78
91
13

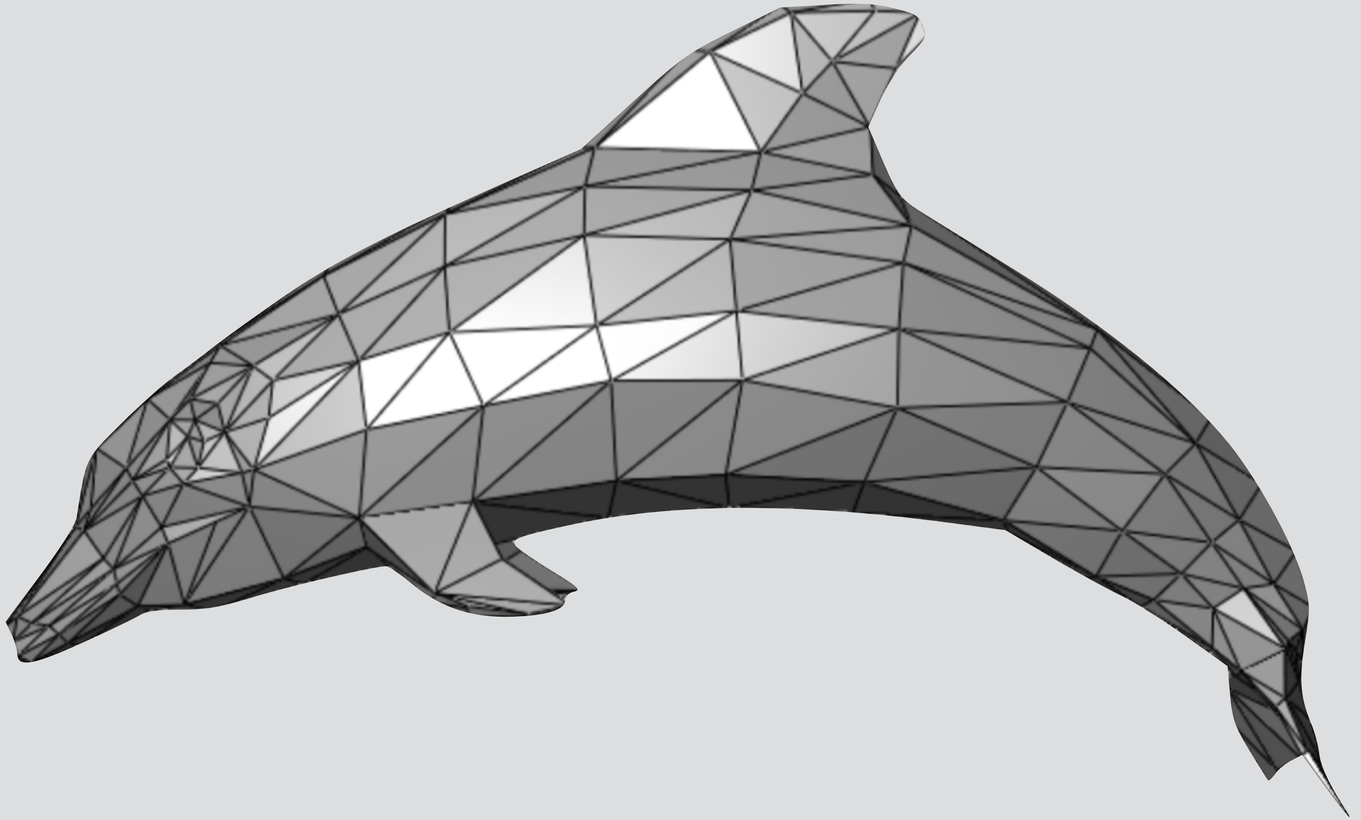

Voodoo3™ 3000 AGP
16MB Ultra-Hochgeschwindigkeits-2D/3D-Beschleunigerkarte mit hoher Auflösung
16MB Ultra hoge-snelheid, hoge-resolutie 2D/3D-versneller
16MB 2D/3D-accelerator med extra hög hastighet och hög upplösning
3dfx™
7 Millionen Triangle/Sek. | 333 Megatexels/Sek. | TV/S-Videoausgang | 3D-Spiele mit 60 fps.
7 miljoen triangles/sec. | 333 megatexels/sec. | TV/S-Video Out | 3D-games met 60 fps.
7 miljoner polygoner/sekund | 333 megatexlar/sekund | TV/S-videoutgång | 3D-spel med 60 bilder per sekund
FREE SOFTWARE INSIDE
Kostenlos in der Verpackung!
Gratuitement à l'intérieure!
Juegos gratis Adentro!
Unreal FULL VERSION
FREE* UPGRADE to Unreal Tournament
*Plus S&H (Details inside)
FULL VERSION FIFA 99
Descent FREESPACE SPECIAL EDITION
MULTILINGUAL MANUAL INCLUDED
Deutsch • Nederlands • Svenska
Voodoo3 3000 AGP
3dfx


133fps
20
0
100
100

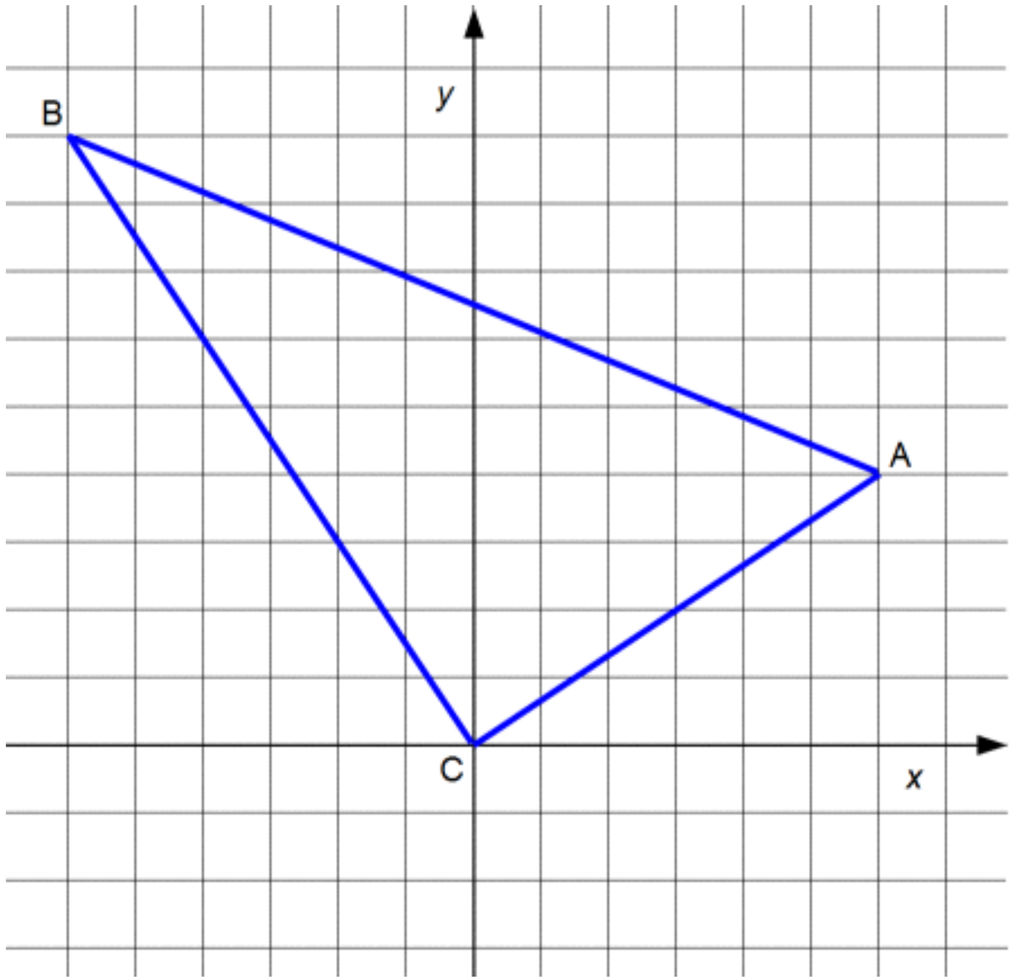
The realtime rendering pipeline.

Rendering **polygons** to a **pixel framebuffer**.

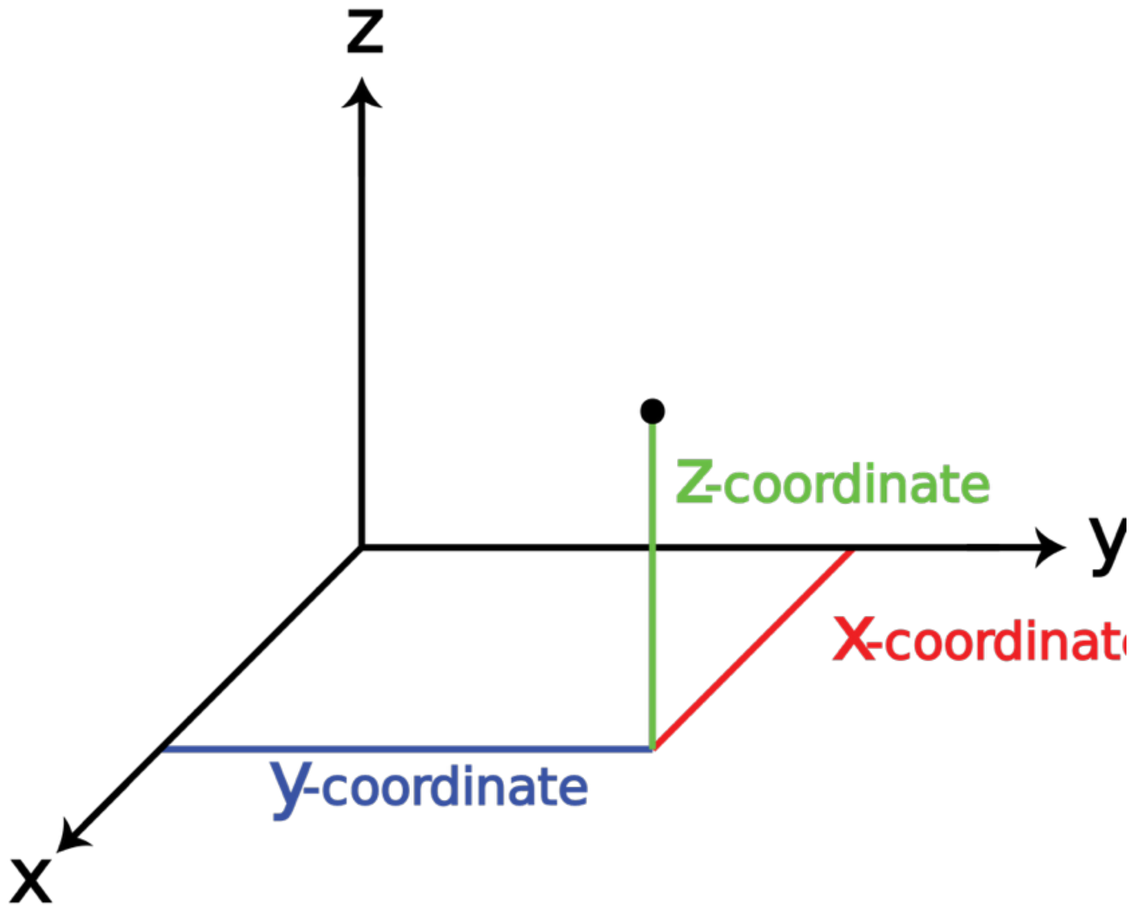
An object is a collection of **triangles**.



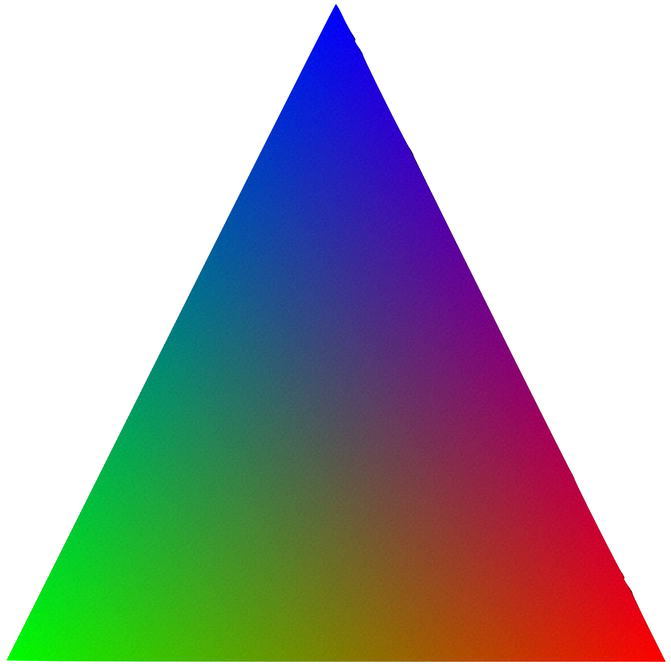
Each triangle is defined by 3 vertices.



Each vertex is described by **(X, Y, Z)** coordinates.



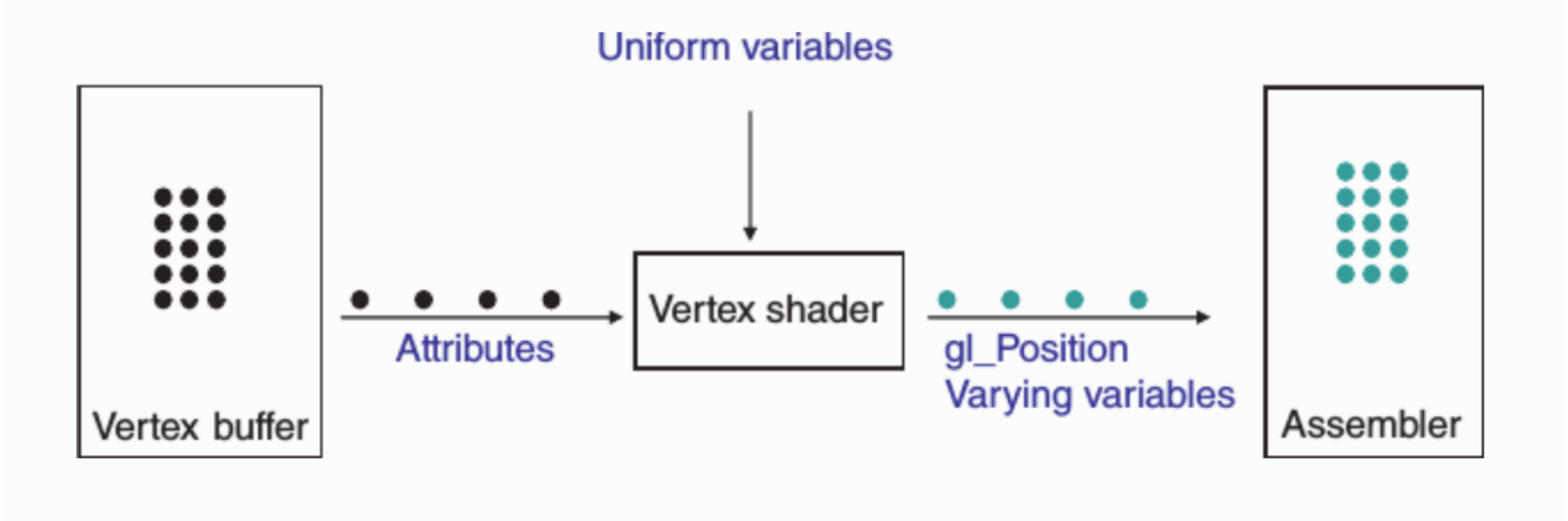
Other information can be attached to each vertex,
such as vertex color and texture coordinate information.



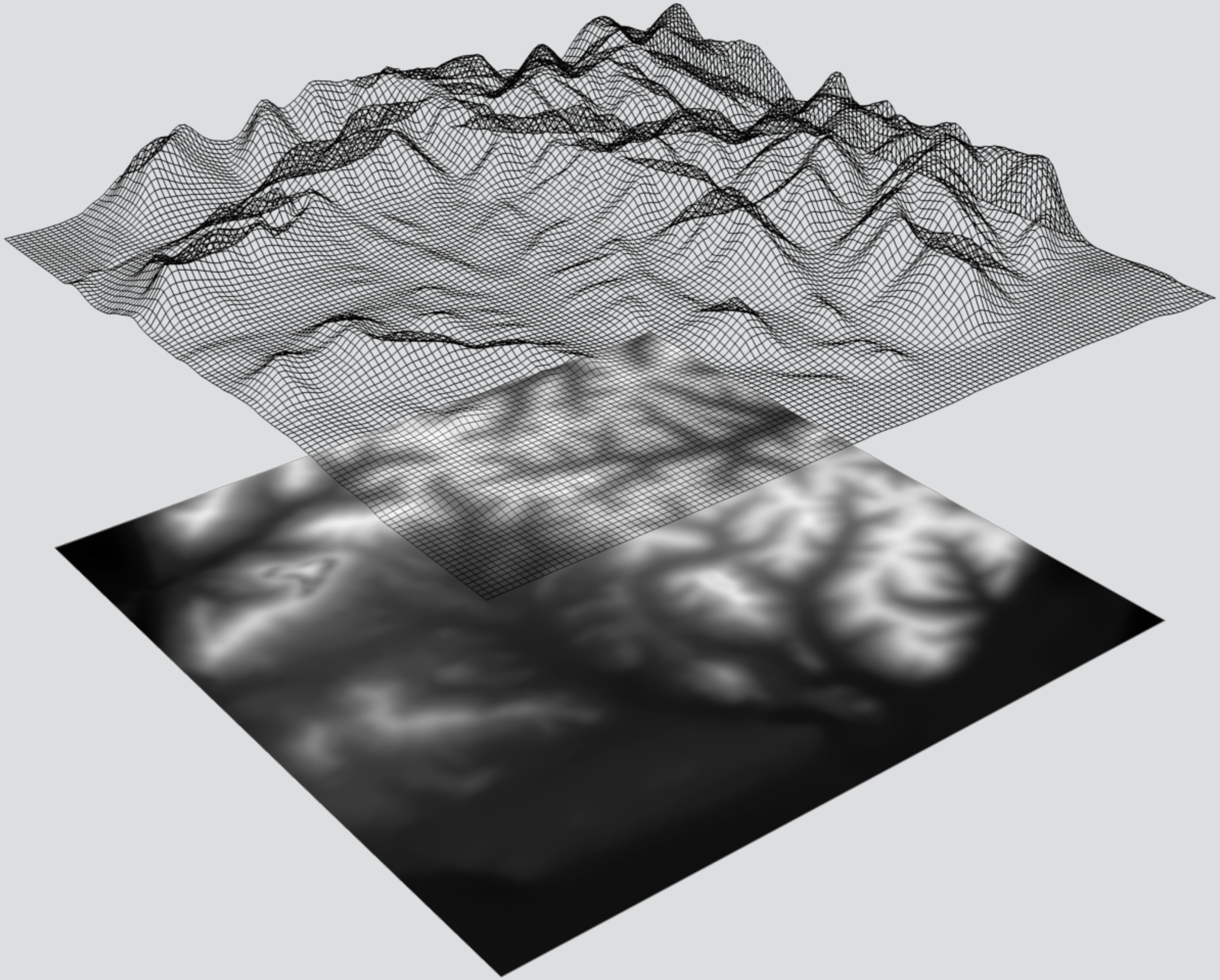
This data is called attribute data.

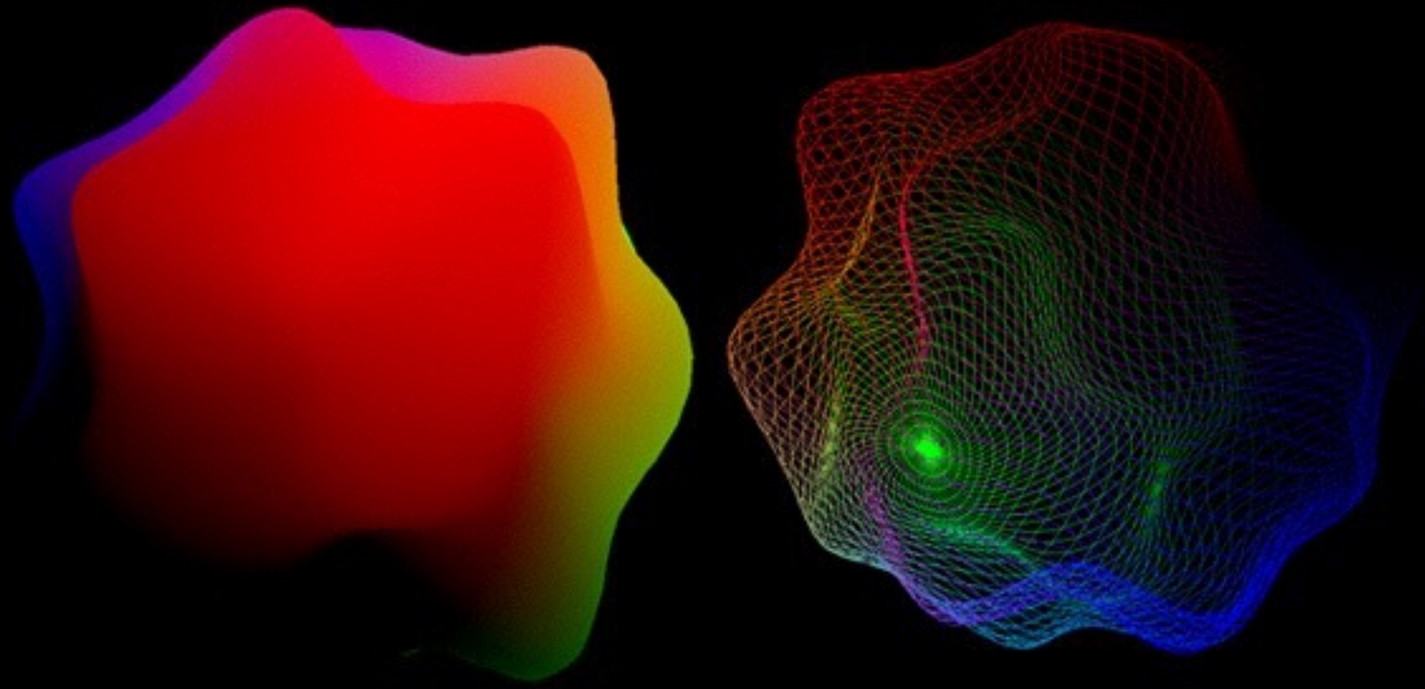
- An object is a collection of **triangles**.
- Each triangle is defined by 3 vertices.
- Each vertex is described by **(X, Y, Z)** coordinates.
- Other information can be attached to each vertex, such as vertex color and texture coordinate information.
- This data is called **attribute data**.

Drawing the triangles with a **draw call**.

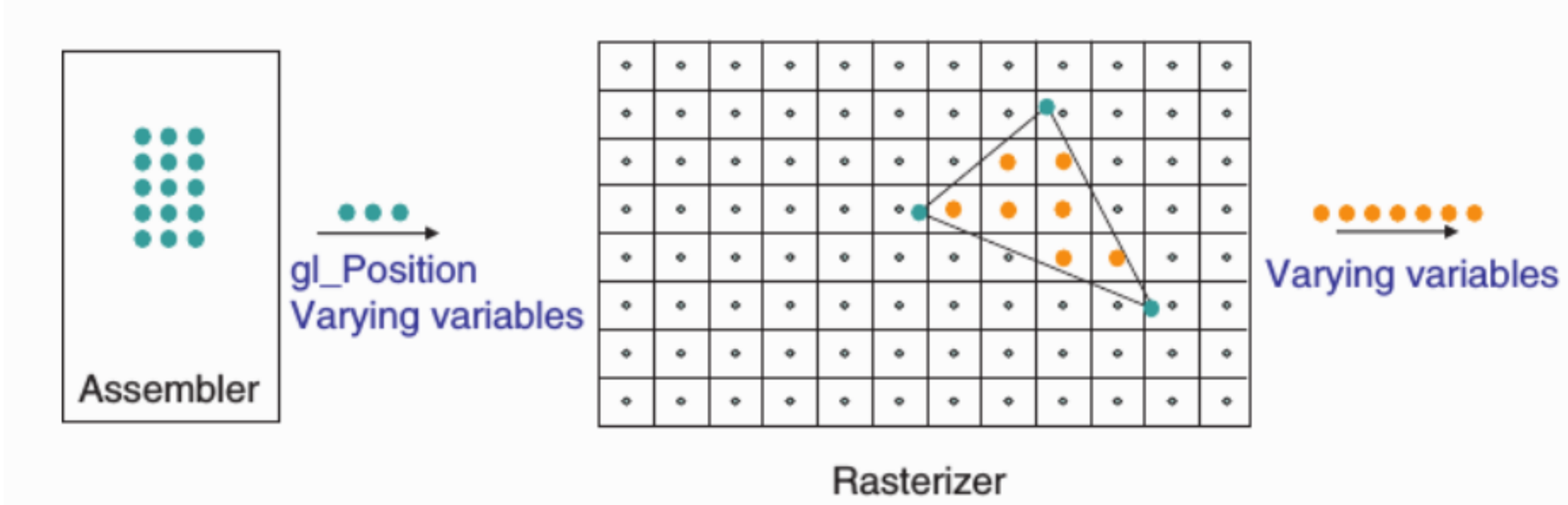

Uniform variables
Vertex buffer
Attributes
Vertex shader
gl_Position
Varying variables
Assembler

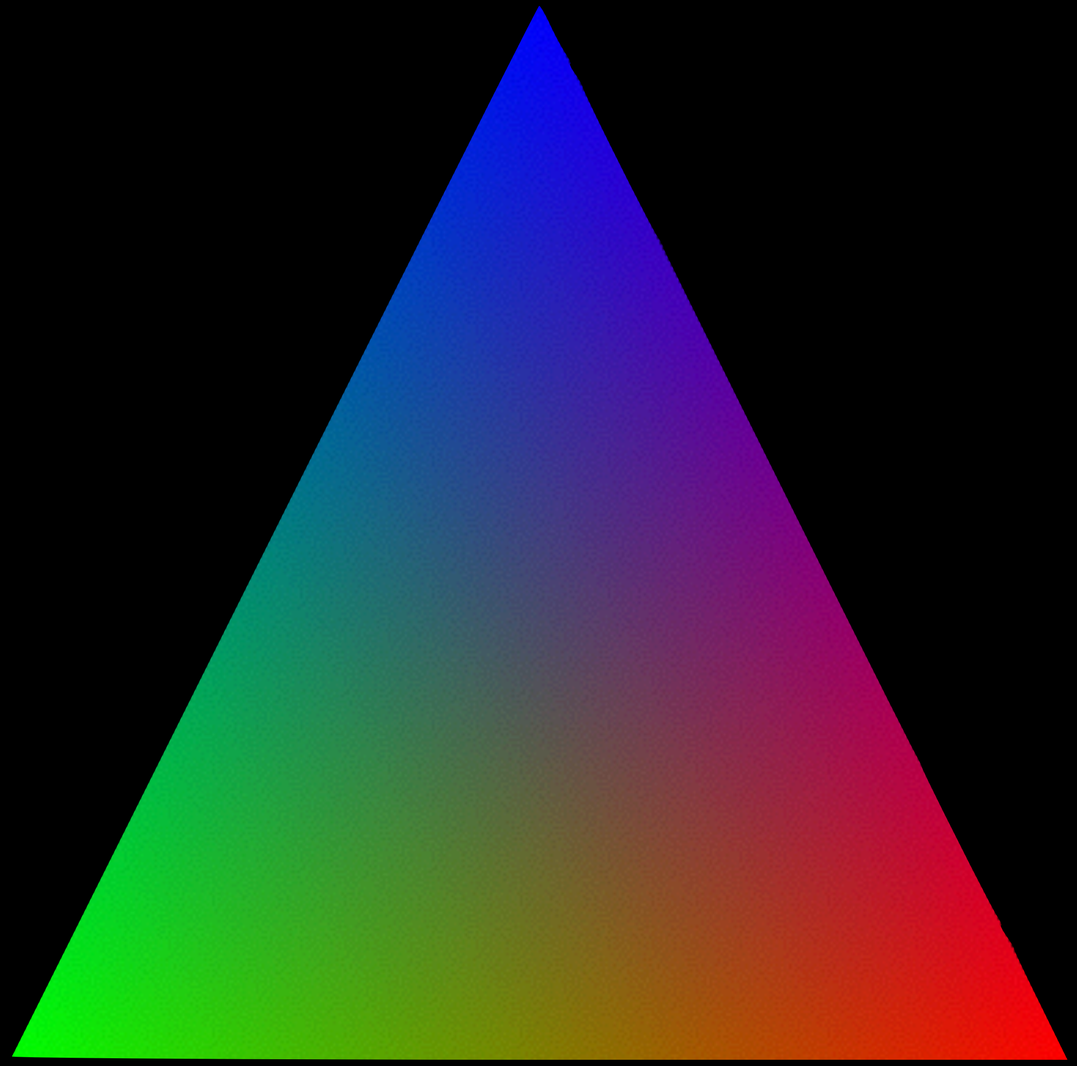
- Each vertex is sent to the **vertex shader** as a set of **attributes**.
- Other data is passed to the vertex shader using **shader uniforms**.
- Typically the vertex shader transforms vertices to their **final positions on the screen**.



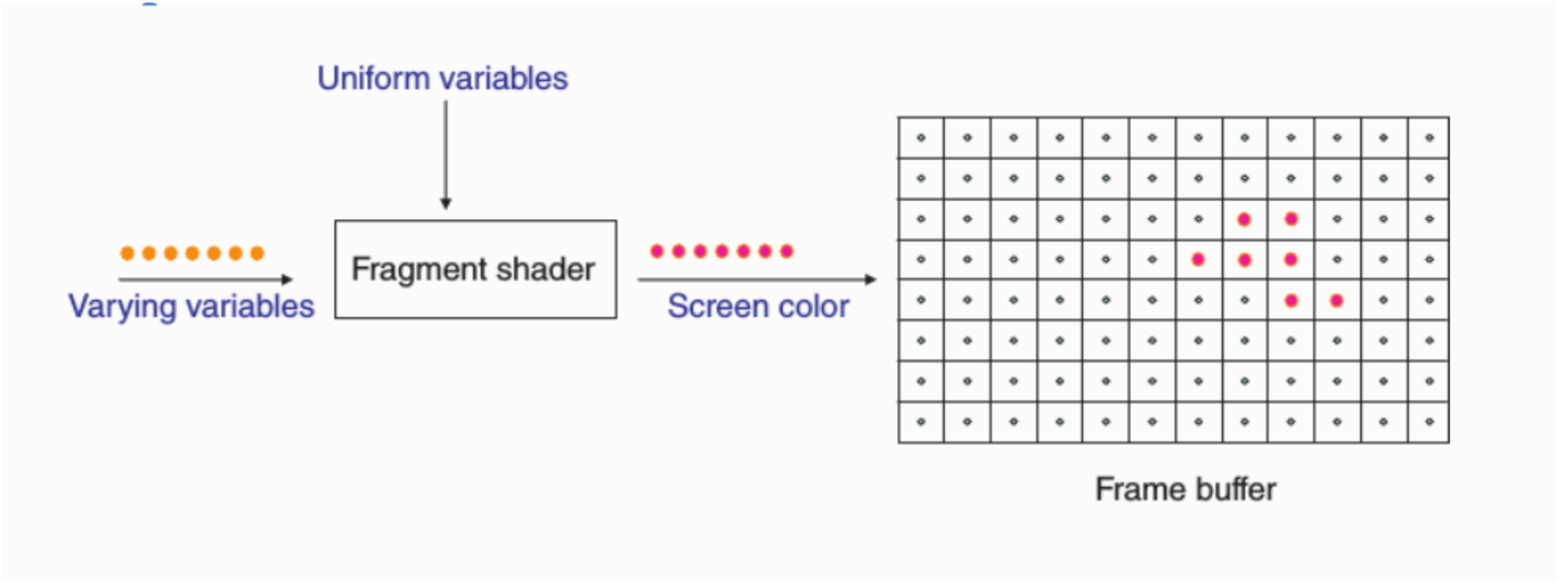


Rasterization


Assembler
gl_Position
Varying variables
Rasterizer
Varying variables

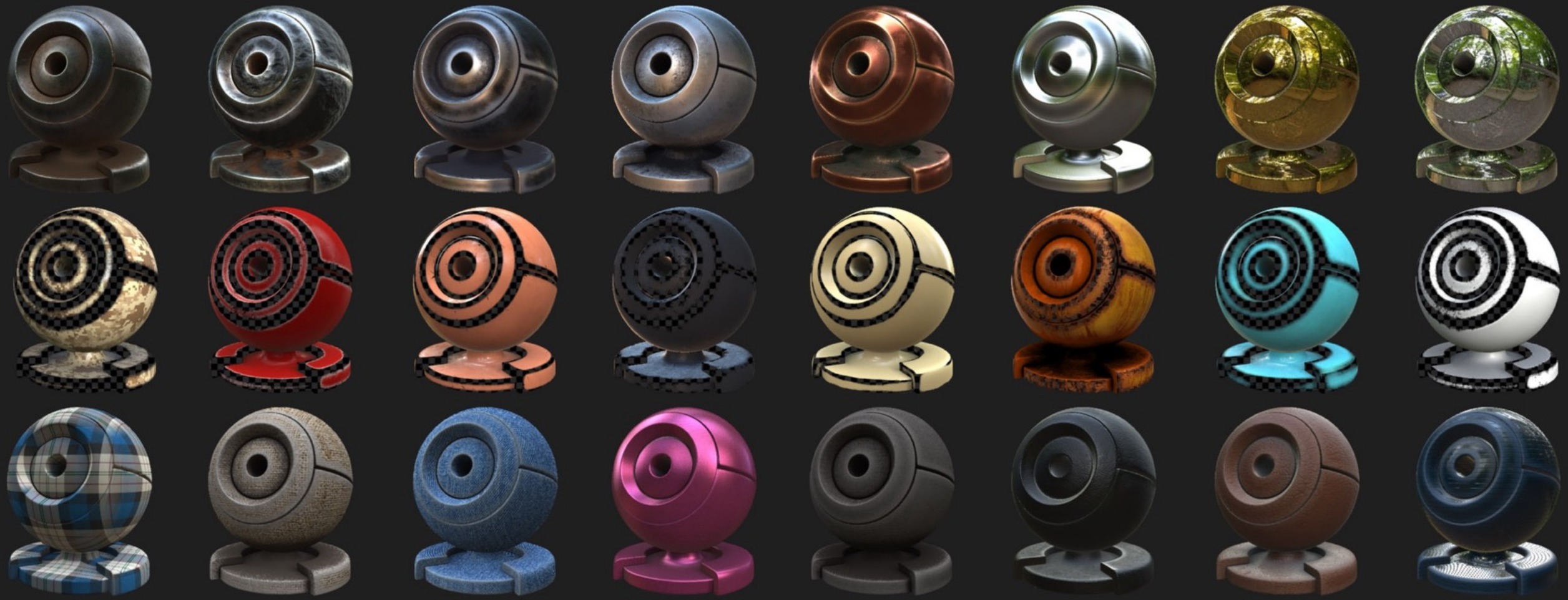


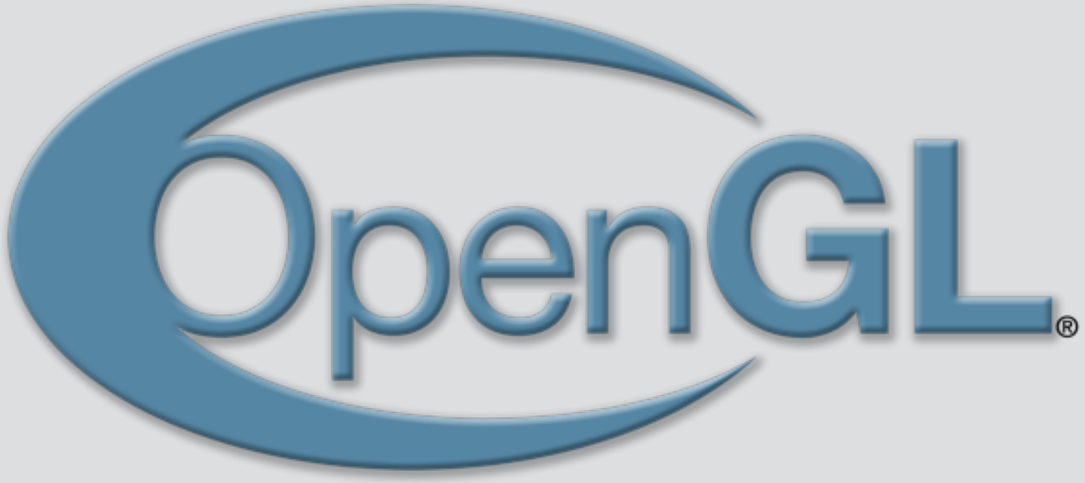
- 3 vertices for each triangle are collected by **the assembler**.
- Each triangle’s vertices are positioned within the window.
- Triangles are rasterized.
- Which pixels are inside the triangle?
- **Varying variable data** (color, normal, texture coordinates) are appropriately interpolated for each pixel.


Uniform variables
Varying variables
Fragment shader
Screen color
Frame buffer

Final fragment output.

- Information for each dot is sent through a **fragment shader** that you write and load into OpenGL.
- Figures out final output color of the pixel from the interpolated data.
- Outputs a color for the screen.





OpenGL and GLSL

GLUT

Assignment 0

Checkout the github repo, build and run the template project.