INTRODUCTION TO

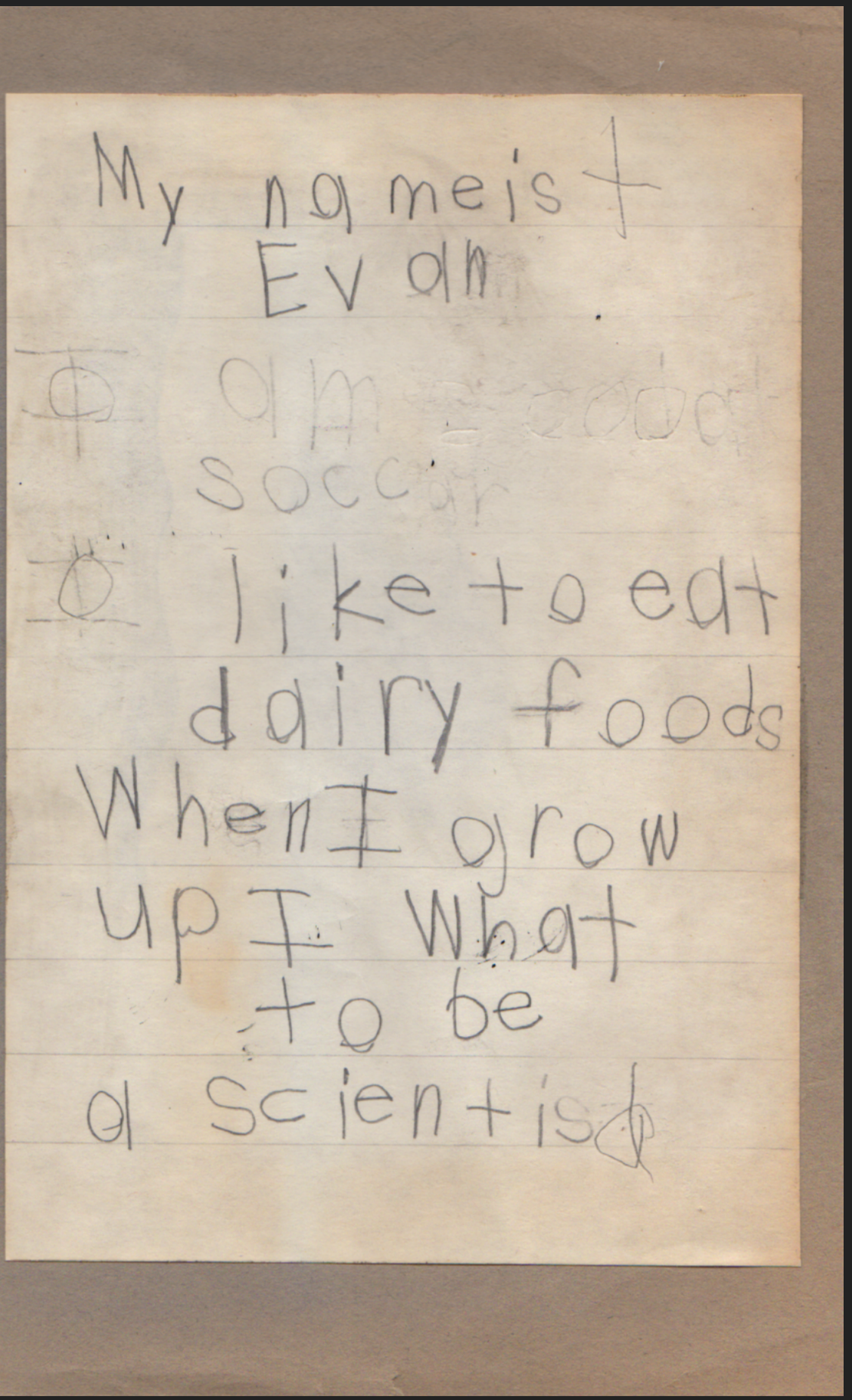
COMPUTER GRAPHICS

ABOUT ME

- Evan Wies
- Software Engineer, programming for >35 years
- Entrepreneur, started 3 companies
- MIT Graduate, Researcher, Alumni Interviewer
- Open Source Contributor (github.com/neomantra)
- @neomantra evan@neomantra.net

ABOUT ME – TECHNOLOGY

- Majored in “Brain and Cognitive Science” and “Electrical Engineering and Computer Science”
- Vision and Haptics Research (3 patents)
- Virtual Reality (VR) and Medical Device Research
- Video Game Startup
- Algorithmic Trading (Fast Data, Big Data)
- Market Data Visualization on Web/Mobile



CAREERS

- Scientist
  They develop theories and work to verify them, expanding human knowledge.
- Engineer
  They leverage knowledge to create solutions to problems.
- Developer / Programmer / Software Engineer
  They create software components and weave them into systems, services, and apps.
- Game Programmer
  They combine art and technology to create magical experiences.

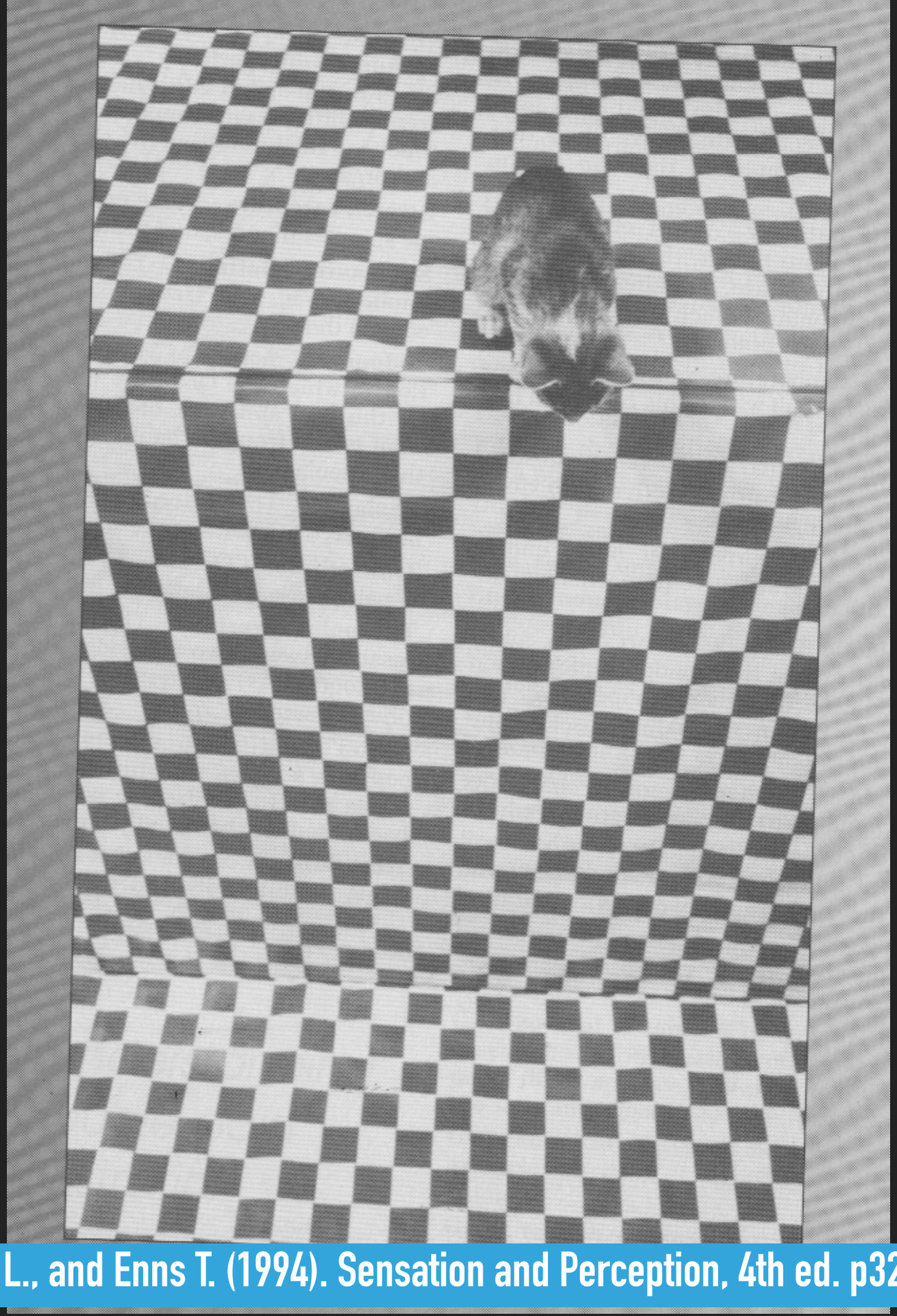
COMPUTER GRAPHICS

- **"CG"** Images created by computers
- Pervasive in film, television, games

- We will peek behind the curtain of how it works
  - Focusing on the lowest-level and real-time
  - How it is principled in Geometry
  - Track the hardware advances that shape our future

VISUAL PERCEPTION

- To create engaging environments (e.g. games),
  we need to understand how humans experience the world



Coren S., Ward L., and Enns T. (1994). Sensation and Perception, 4th ed. p326. Harcourt Brace

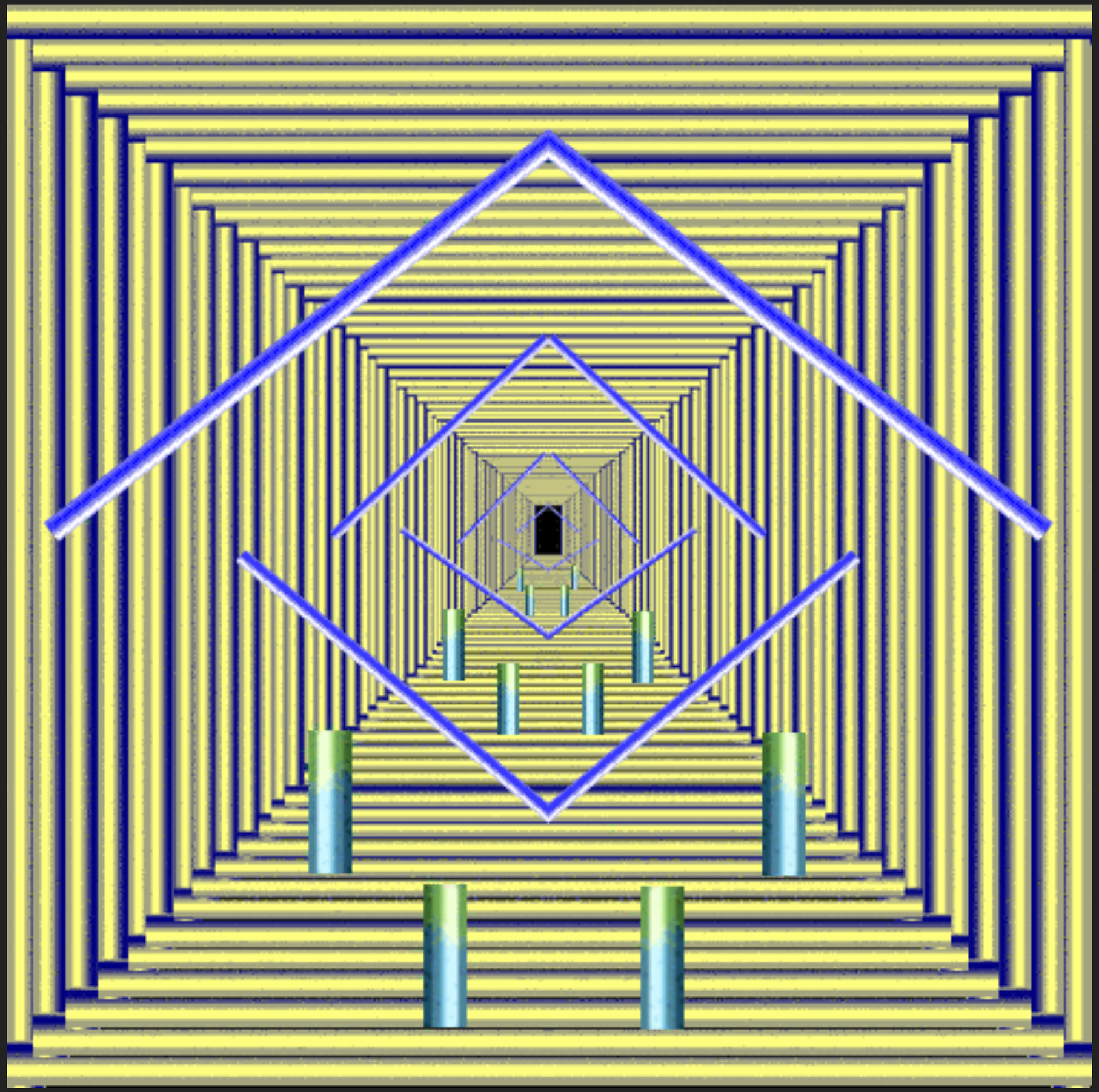
VISUAL PERCEPTION – BRAIN AND COGNITIVE SCIENCE

- Brain Science = Bottom-Up
  - Understanding the physical, from anatomy (structures) and physiology (function and relationships) of our electro-chemical-mechanical system
- Cognitive Science = Top-Down
  - Understanding thought - perception, reading, memory, problem-solving, task planning, motion planning

VISUAL PERCEPTION

- To create engaging environments (e.g. games),
  we need to understand how humans experience the world
- Visual Perception
- Depth Cues - Perspective, Parallax
- Color and Lighting
- Apparent Motion

ANDREA MANTEGNA, FRESCO, CAMERA DEGLI SPOSI, DUCAL PALACE, MANTUA, C. 1470.



https://giphy.com/gifs/xenosef-3o85xEaTki0R0QYpNu

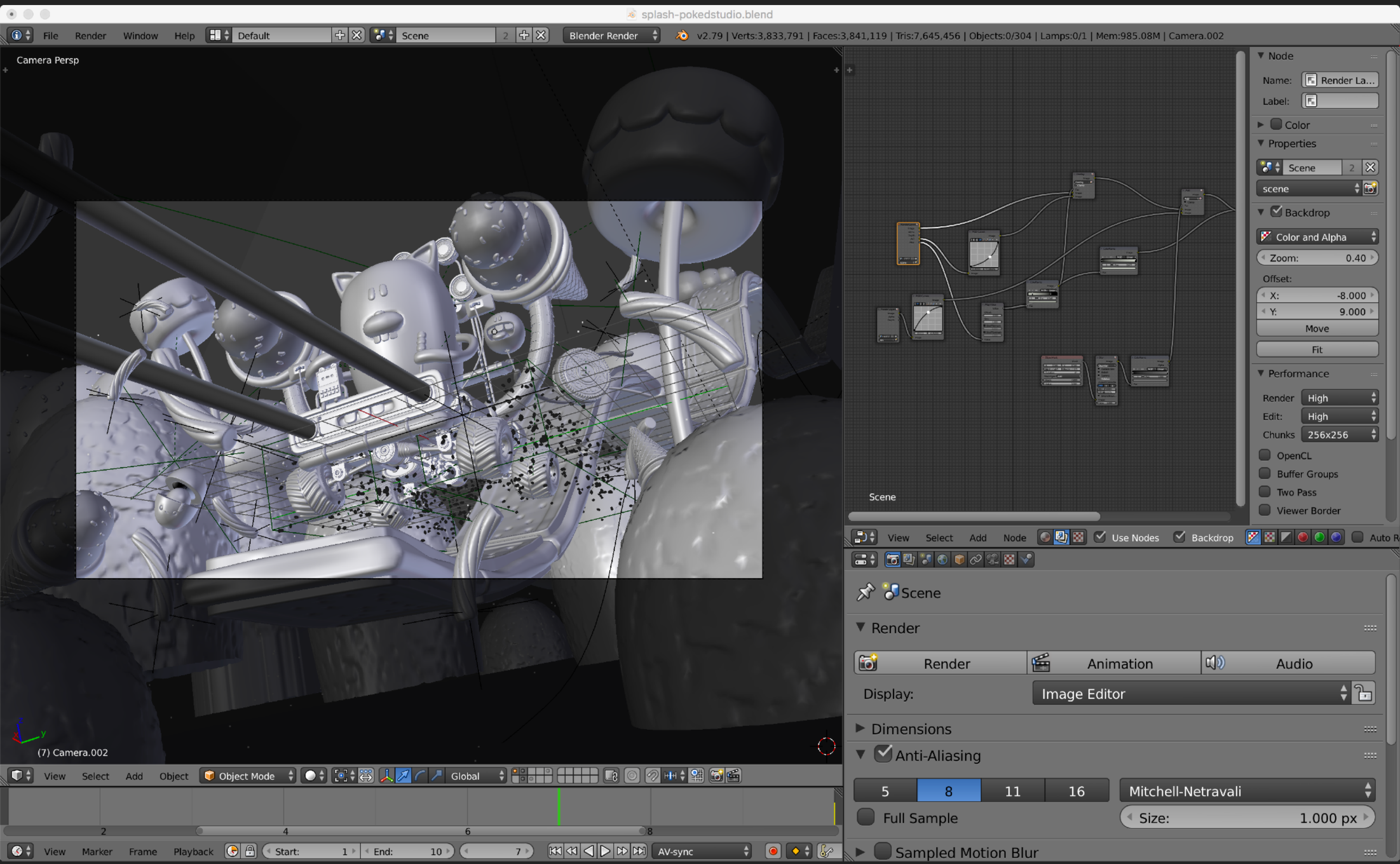
VISUAL PERCEPTION

- To create engaging environments (e.g. games),
  we need to understand how humans experience the world
- Visual Perception
  - Depth Cues - Perspective, Parallax
  - Color and Lighting
  - Apparent Motion
- Psychophysics
  - Temporal Fusion

RENDERING

- Taking a “scene” filled with “models”, lights, and effects and producing an image
- Scenes and models are created in 3D modeling software

BLENDER MODELING TOOL

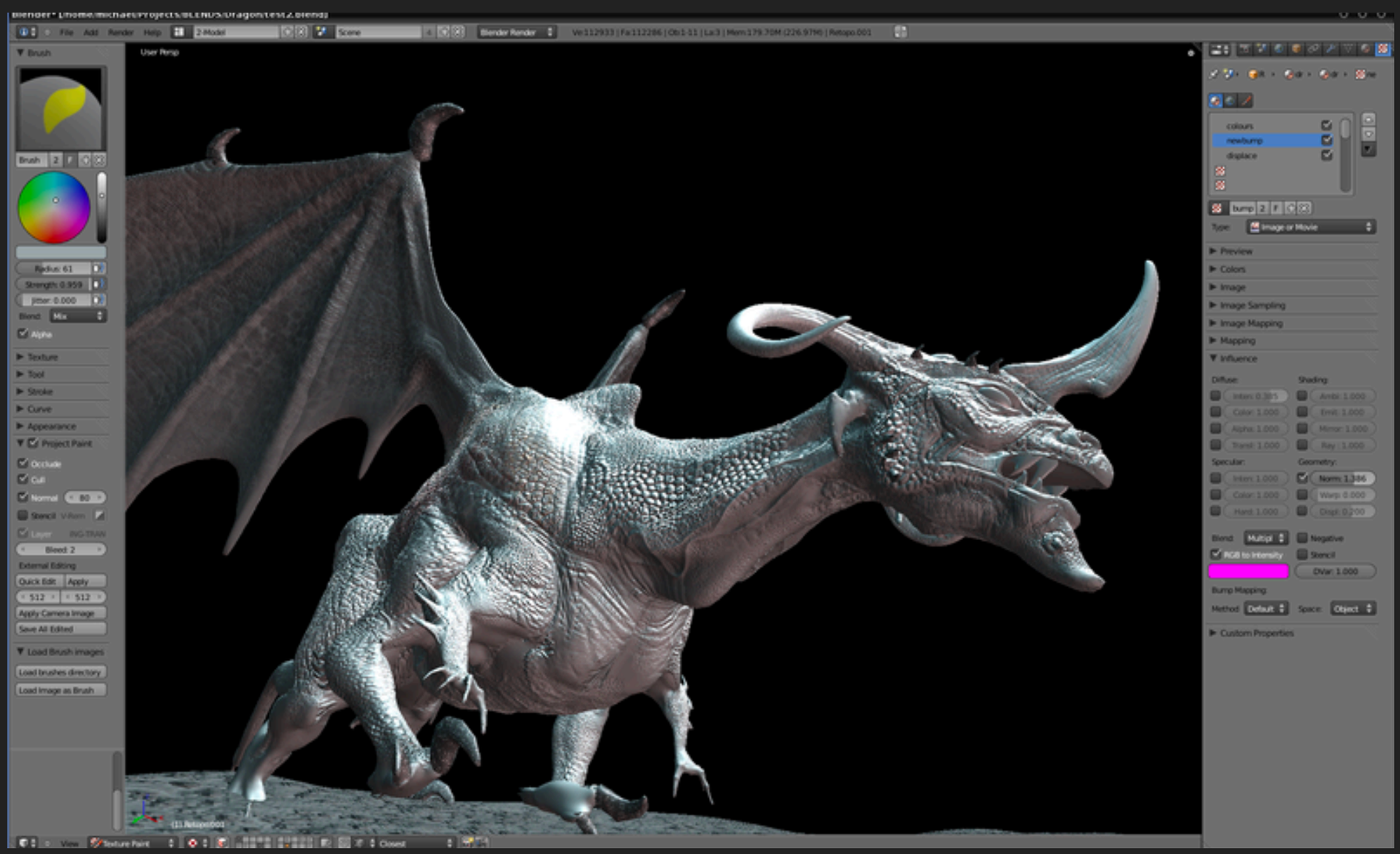


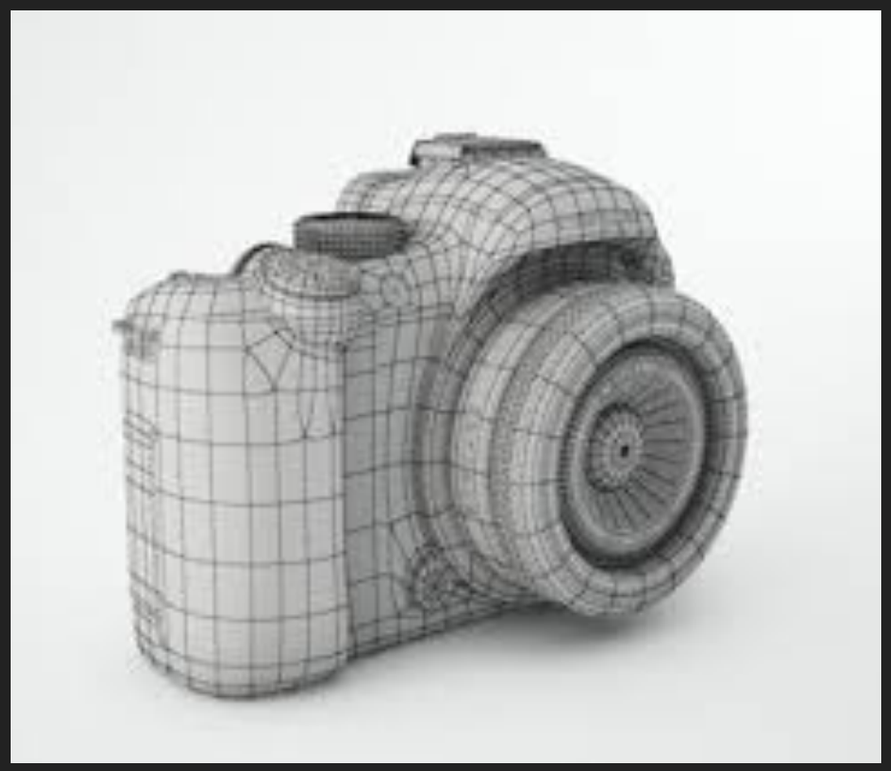
https://www.blender.org/download/demo-files/ "2.77 Racing Car" by Pokedstudio

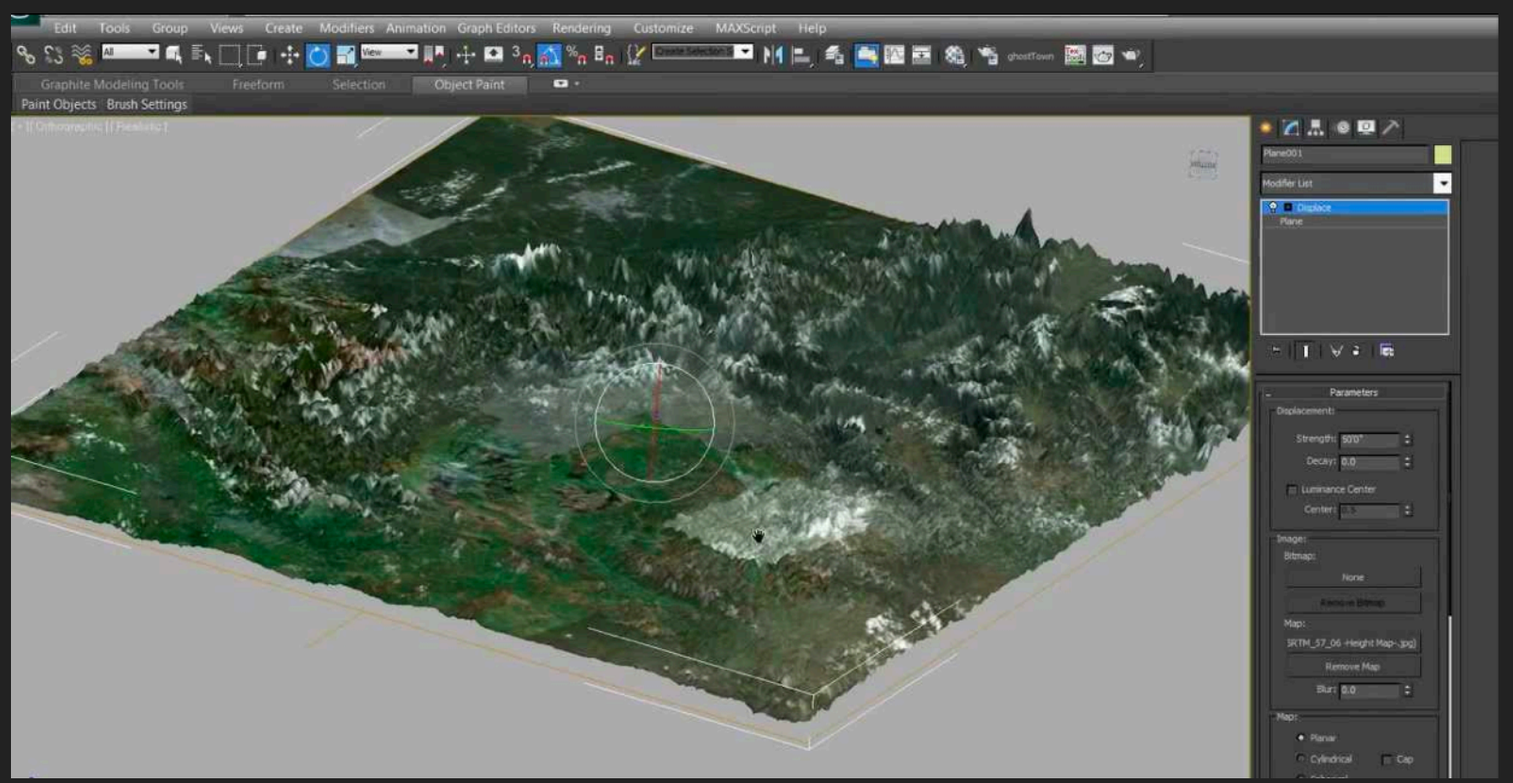
RENDERED SCENE – 21 MINUTES

https://www.blender.org/download/demo-files/ “2.77 Racing Car” by Pokedstudio

MODELS







MODELS - COLOR

- Color perception and representation is a deep field
- CG usually represents color using:
  Red, Green, Blue (RGB)
  Values between 0.0 and 1.0
  (0,0,0) is black, (1,1,1) is white, (0.5, 0.5, 0.5) is grey
  (1.0, 0.98, 0.32) is yellow
- Alpha for transparency (RGBA)
  0 is transparent, 1 is opaque

MODELS – TEXTURE MAPS

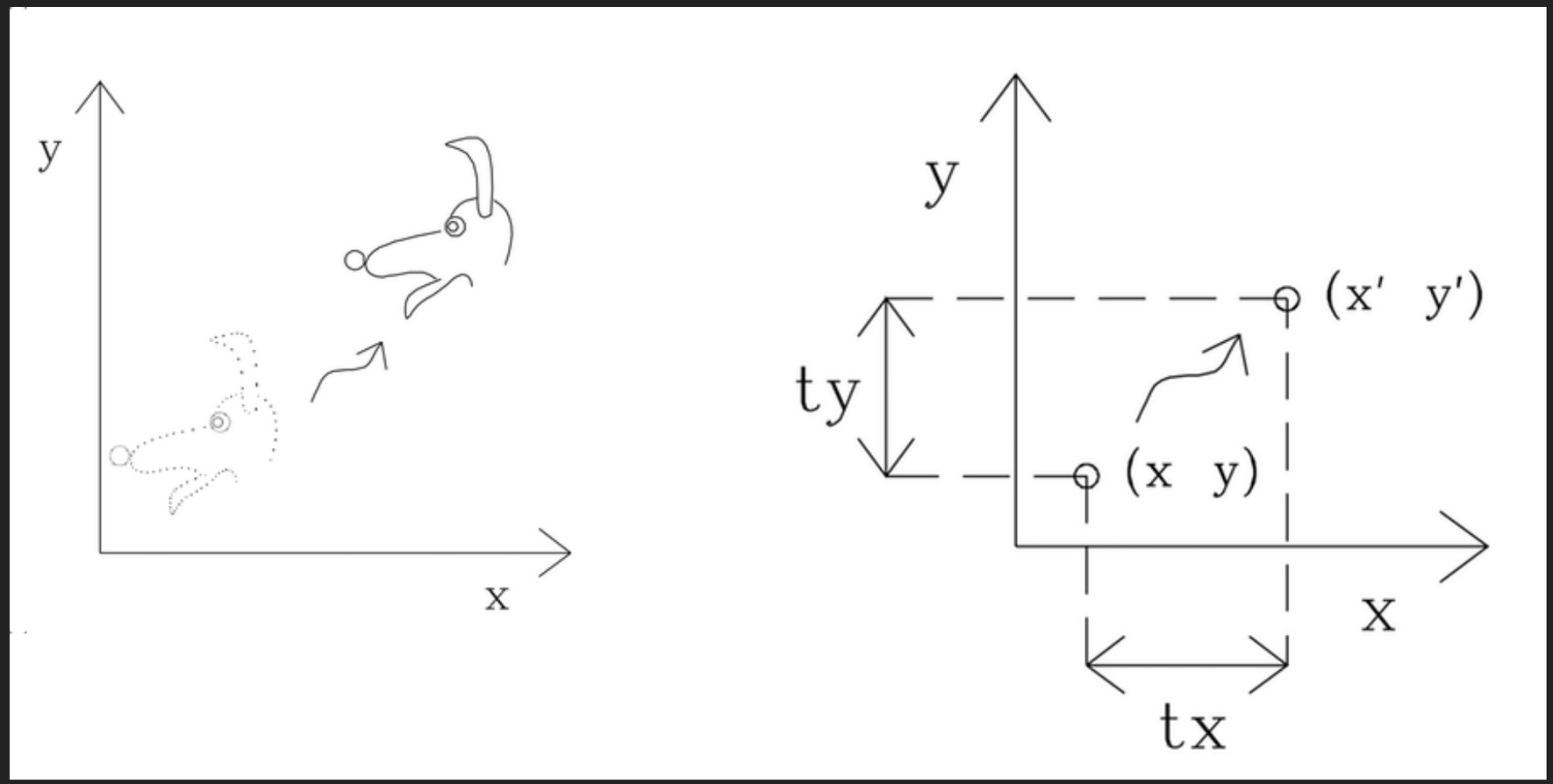
- Textures Maps are images (drawn, captured, procedural)
- Referred to with “texture coordinates” in “uv space”

http://www.bobotheseal.com/personal_work/textures/engineer-final02.jpg

SCENES

- Scenes are compositions of models, materials, lighting, special effects, physics, and programmed logic (e.g. AI)
- Models are placed in the scene through geometric transformations:
  - Translation
  - Scaling
  - Rotation

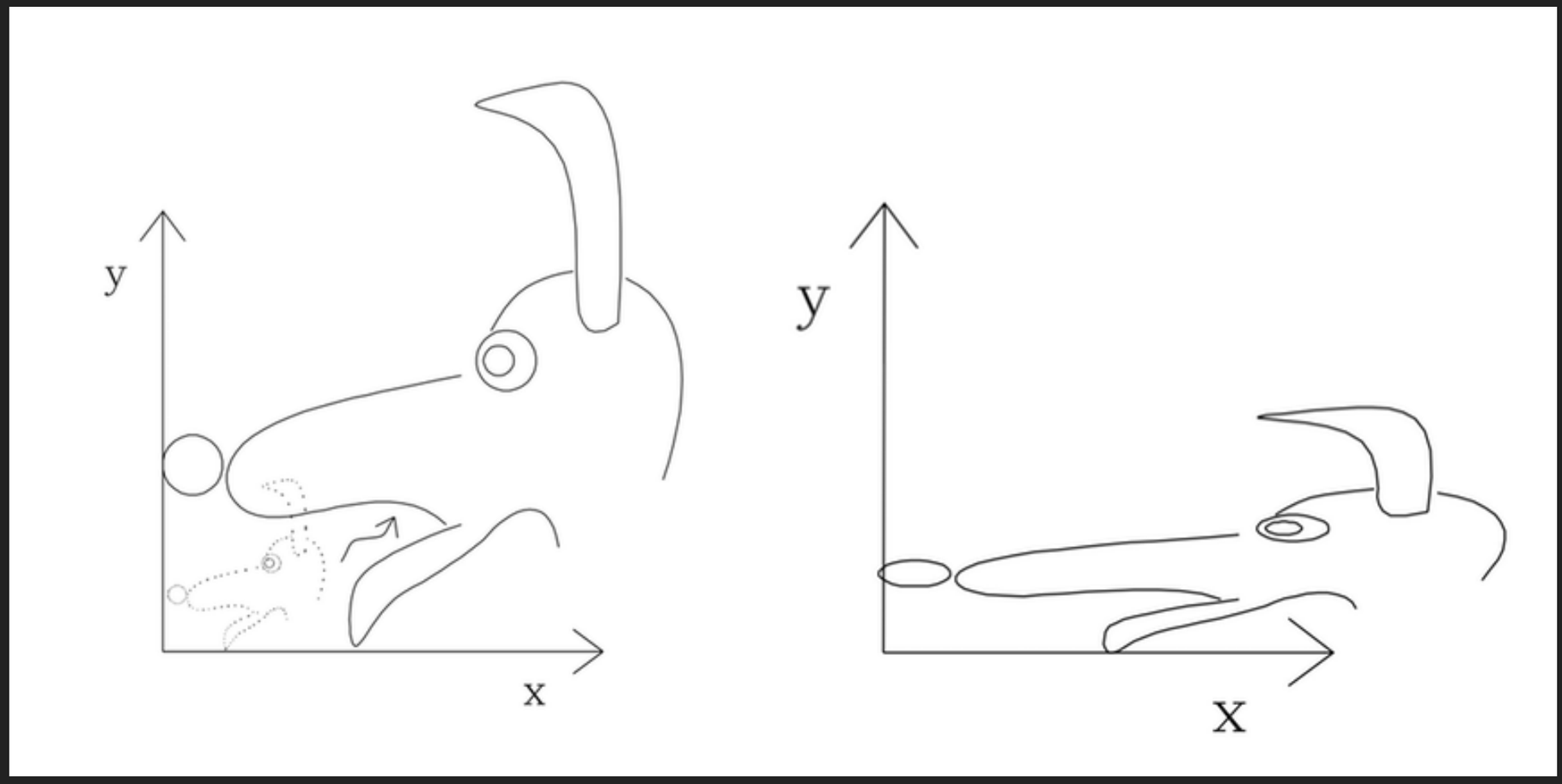
SCENES – TRANSFORMS: TRANSLATION



$$x' = x + t_x$$
$$y' = y + t_y$$

http://www.dis.uniroma1.it/~degiacom/didattica/informatica_grafica/geom_sn.pdf

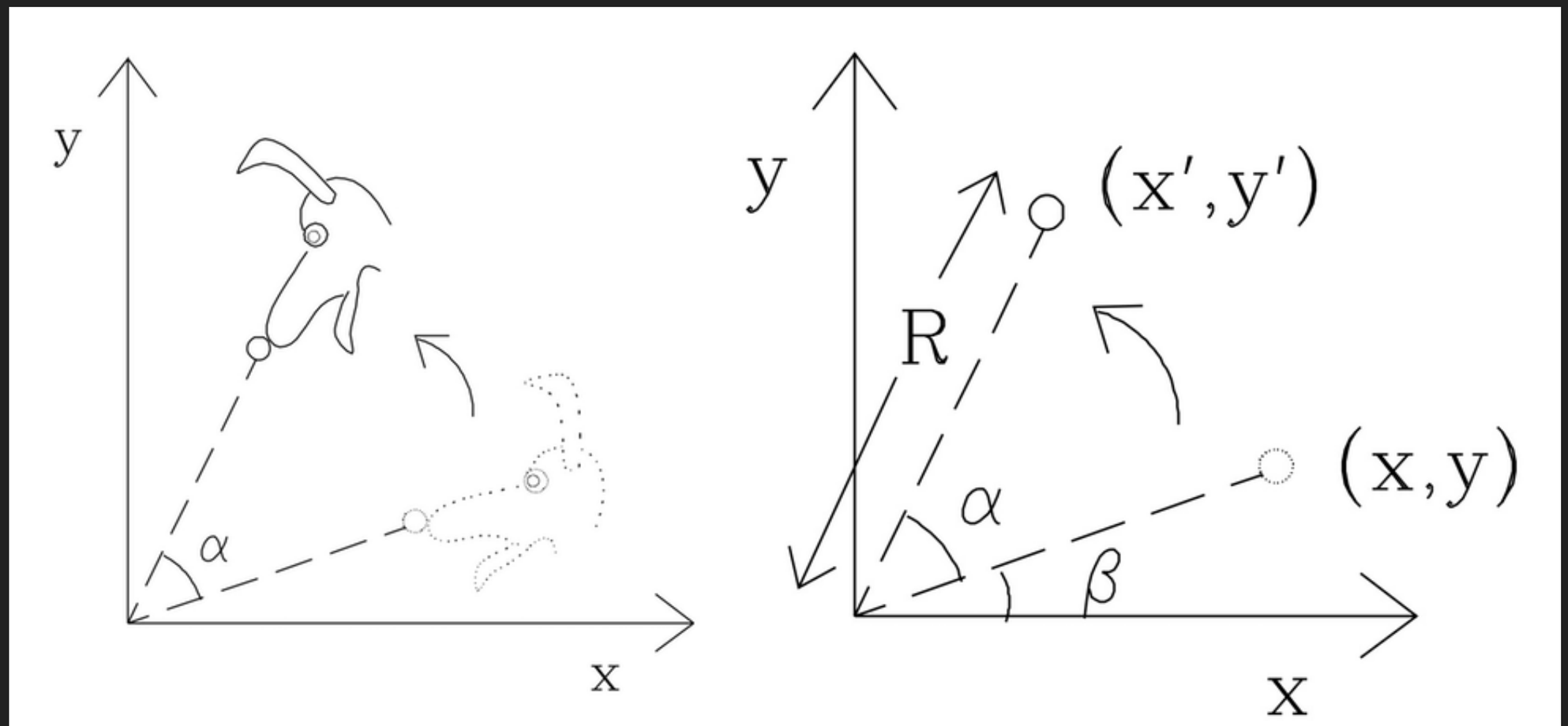
SCENES – TRANSFORMS: SCALING



$$x' = x * s_x$$
$$y' = y * s_y$$

http://www.dis.uniroma1.it/~degiacom/didattica/informatica_grafica/geom_sn.pdf

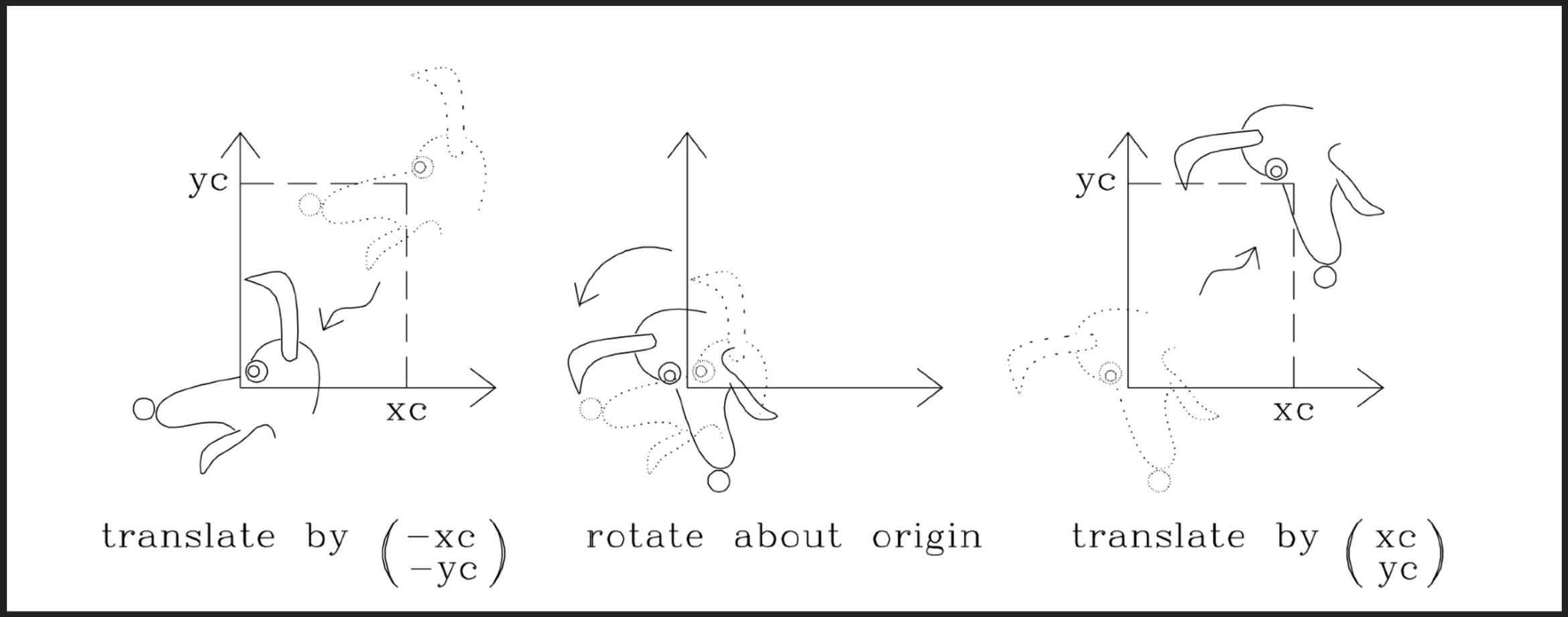
SCENES – TRANSFORMS: ROTATION



$$x' = x * \cos\alpha - y * \sin\alpha$$
$$y' = x * \sin\alpha + y * \cos\alpha$$

http://www.dis.uniroma1.it/~degiacom/didattica/informatica_grafica/geom_sn.pdf

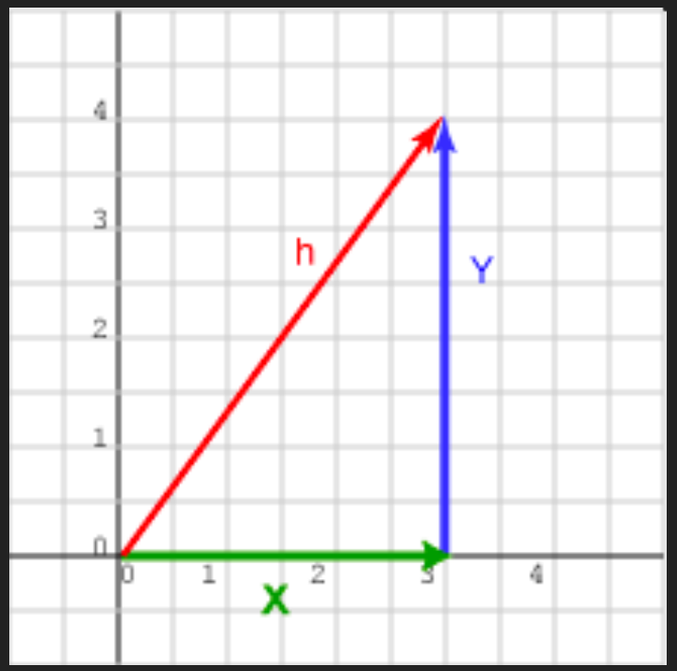
SCENES – TRANSFORMS: COMBINED



http://www.dis.uniroma1.it/~degiacom/didattica/informatica_grafica/geom_sn.pdf

TIMEOUT – ABSTRACTION

- Software Engineering is all about “controlling complexity”
- “Abstraction”
  - Establish a level of simplicity, hiding the details
  - You can freely operate at a level with trust
  - Can go up and down the layers as needed




```
vec2 v = vec2(3.0, 4.0);

float len = sqrt(v.x*v.x + v.y*v.y);

float len = length(v);
```

SCENES – TRANSFORMS: AS MATRICES

- Transformations can be represented with matrices
- Transformations are composed by multiplying the matrices
- Points are transformed by multiplying their vector form by the transformation matrix

```
Matrix3x3 m1;
m1.translate(Vec3(1,1,0));
m1.rotate(180, Vec3(1,0,0));

Matrix3x3 m2;
m2.scale(Vec3(3, 2, 1));

Vec3 v = m2 * m1 * Vec3(1, 1, 0);
```

- Translate $\begin{bmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{bmatrix}$
- Scale $\begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- Rotate $\begin{bmatrix} \cos\alpha & -\sin\alpha & 0 \\ \sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix}$
- Shear $\begin{bmatrix} 1 & a & 0 \\ b & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$

http://www.dis.uniroma1.it/~degiacom/didattica/informatica_grafica/geom_sn.pdf

SCENES – MATRIX MATH

- Matrix Times Vector:

$$\begin{bmatrix} a & b & c & d \\ e & f & g & h \\ i & j & k & l \\ 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} = \begin{bmatrix} ax + by + cz + d \\ ex + fy + gz + h \\ ix + jy + kz + l \\ 1 \end{bmatrix}$$

- Matrix Times Matrix:

$$\begin{bmatrix} a & b & c \\ d & e & f \\ g & h & i \end{bmatrix} \times \begin{bmatrix} j & k & l \\ m & n & o \\ p & q & r \end{bmatrix} =$$

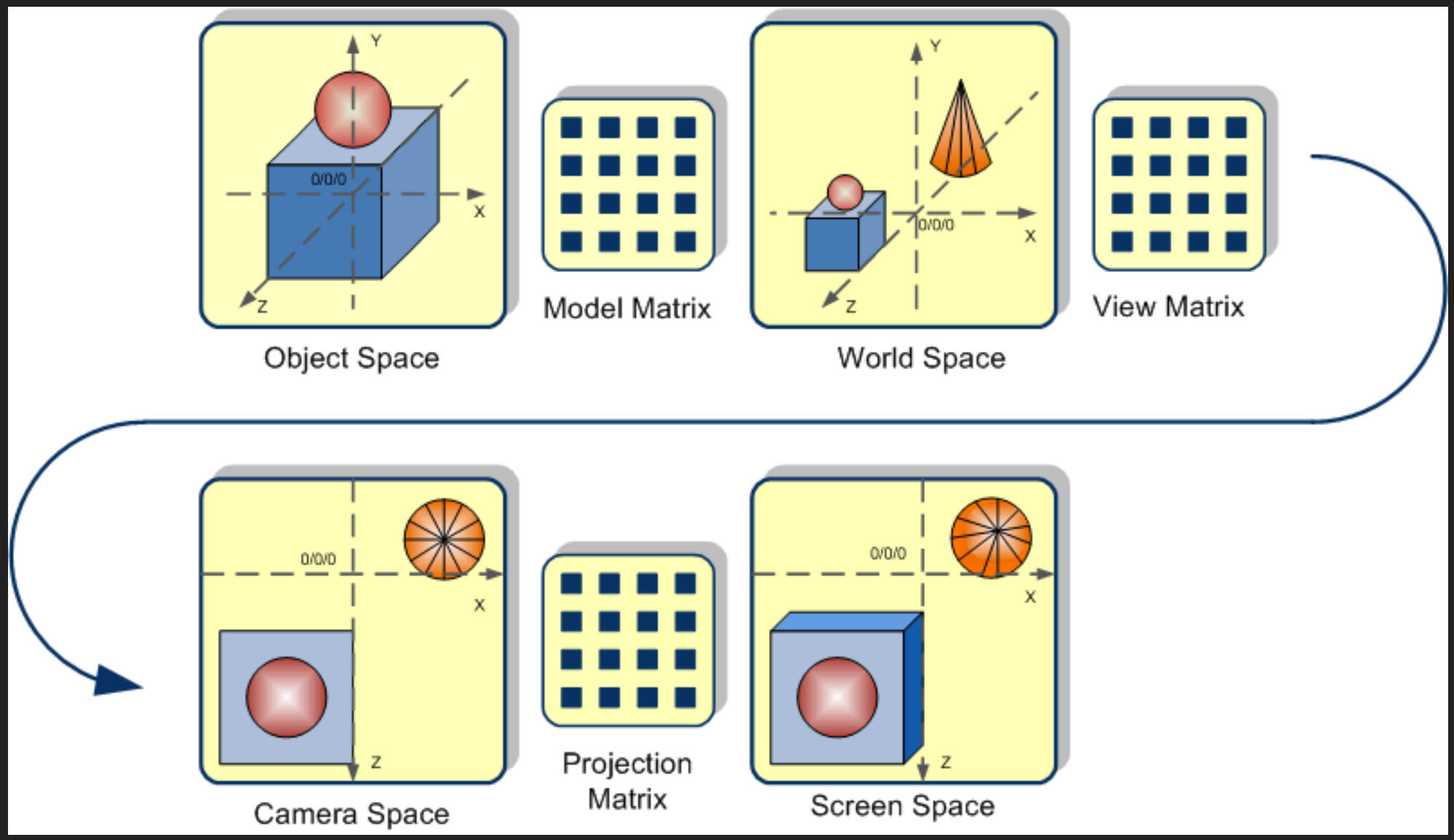
$$\begin{bmatrix} aj + bm + cp & ak + bn + cq & al + bo + cr \\ dj + em + fp & dk + en + fq & dl + eo + fr \\ gj + hm + ip & gk + hn + iq & gl + ho + ir \end{bmatrix}$$

- KEY: It is all ADDITION and MULTIPLICATION!

http://www.c-jump.com/bcc/common/Talk3/Math/Matrices/Matrices.html

SCENES – SPACES

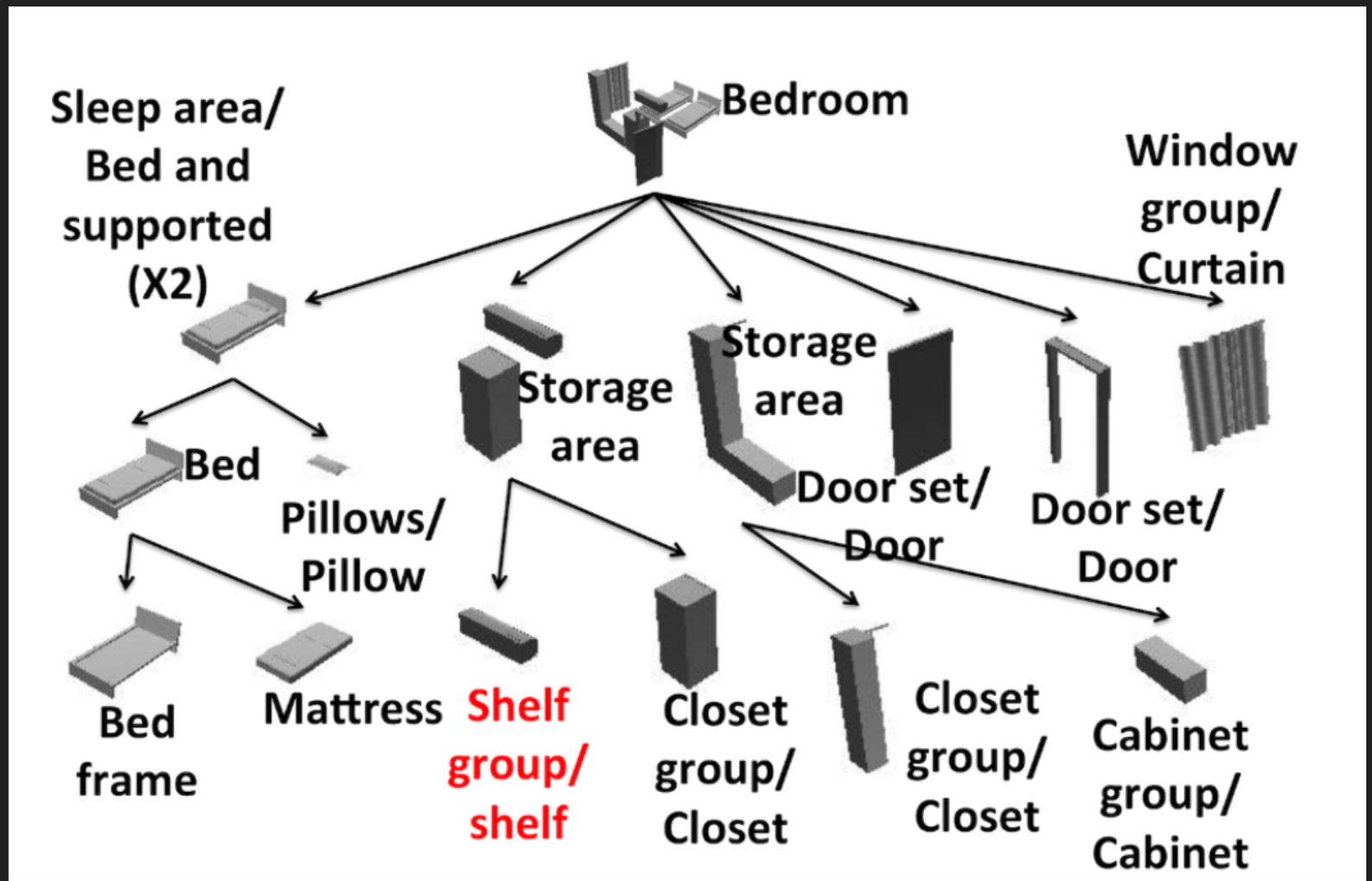
- Objects in 3D scenes exist in spaces:



http://www.c-jump.com/bcc/common/Talk3/Math/Matrices/Matrices.html

SCENE GRAPH

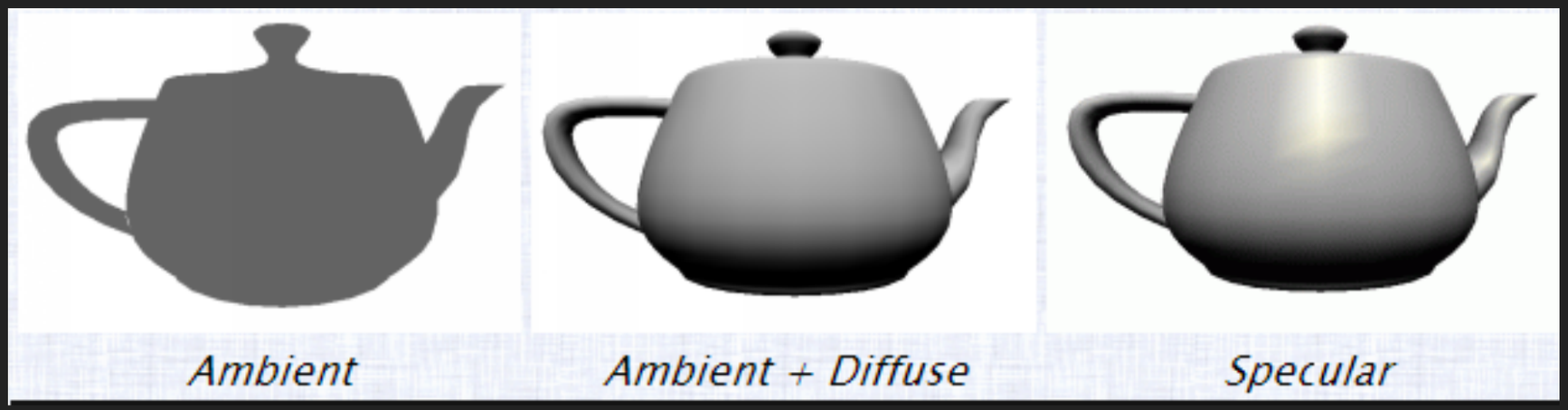
- Graph = “a set of objects that are connected together”
- Directed Acyclic Graph (DAG)…
  There’s an order to it and there are no loops
- For CG, the nodes are models, materials, lights, and transforms
- Renderers traverse the graph to produce a frame of the scene

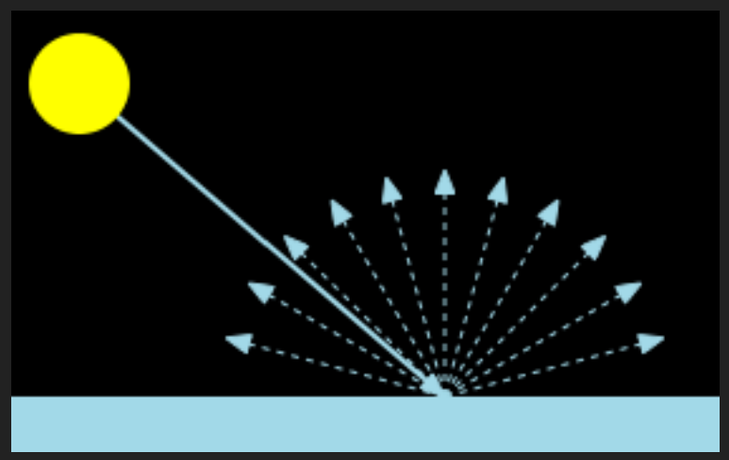


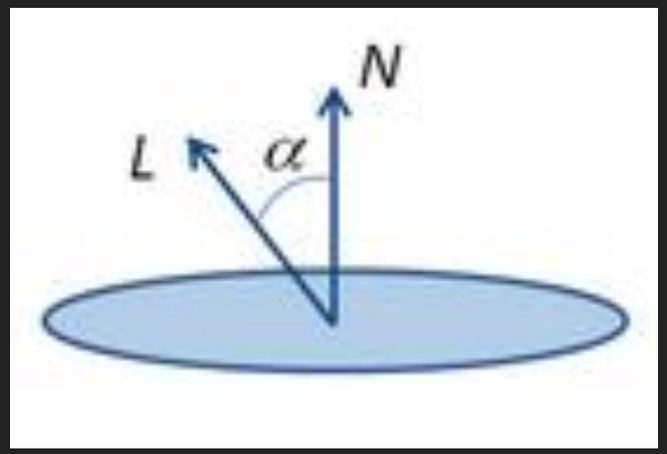
http://www.ctralie.com/Teaching/COMPSCI290/Lectures/7_ImageSources/

LIGHTING MODELS

- We see the world through light reflecting off objects
- Different kinds of lights. We mathematically model lights to get a visual effect.
- The models are usually grounded in physics, but also might be something "good enough".







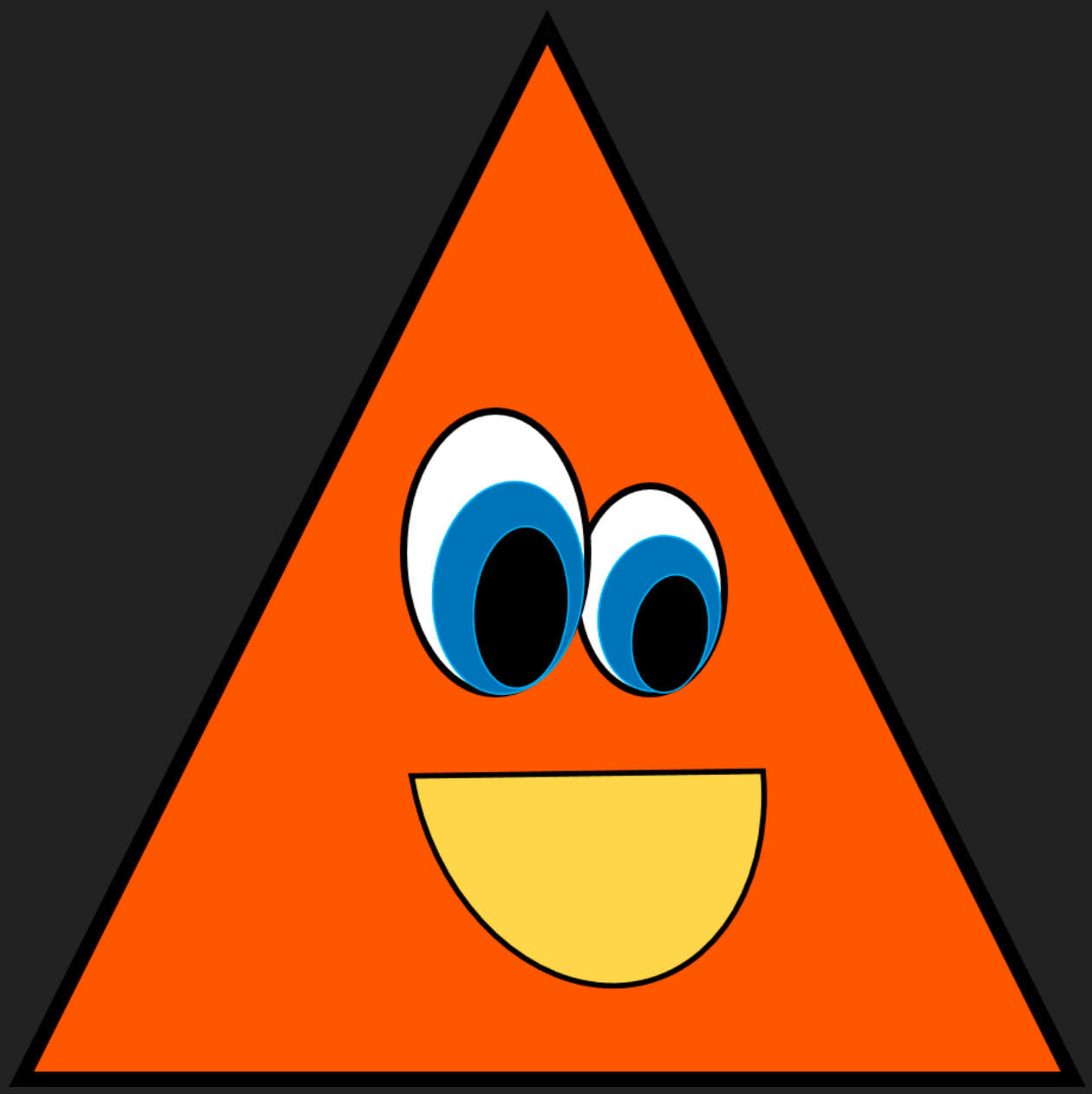

```
Id = max(Kd * max(dot(L,N),0), Ia)

Id: diffuse light intensity
Ia: ambient light intensity
Kd: material diffuse value
L:  incoming light vector
N:  surface normal
```

http://www.c-jump.com/bcc/common/Talk3/OpenGL/Wk06_light/Wk06_light.html

REAL-TIME RENDERING

- We've got models, materials, lighting, transforms...
- How can we experience it at 60 frames per second?!

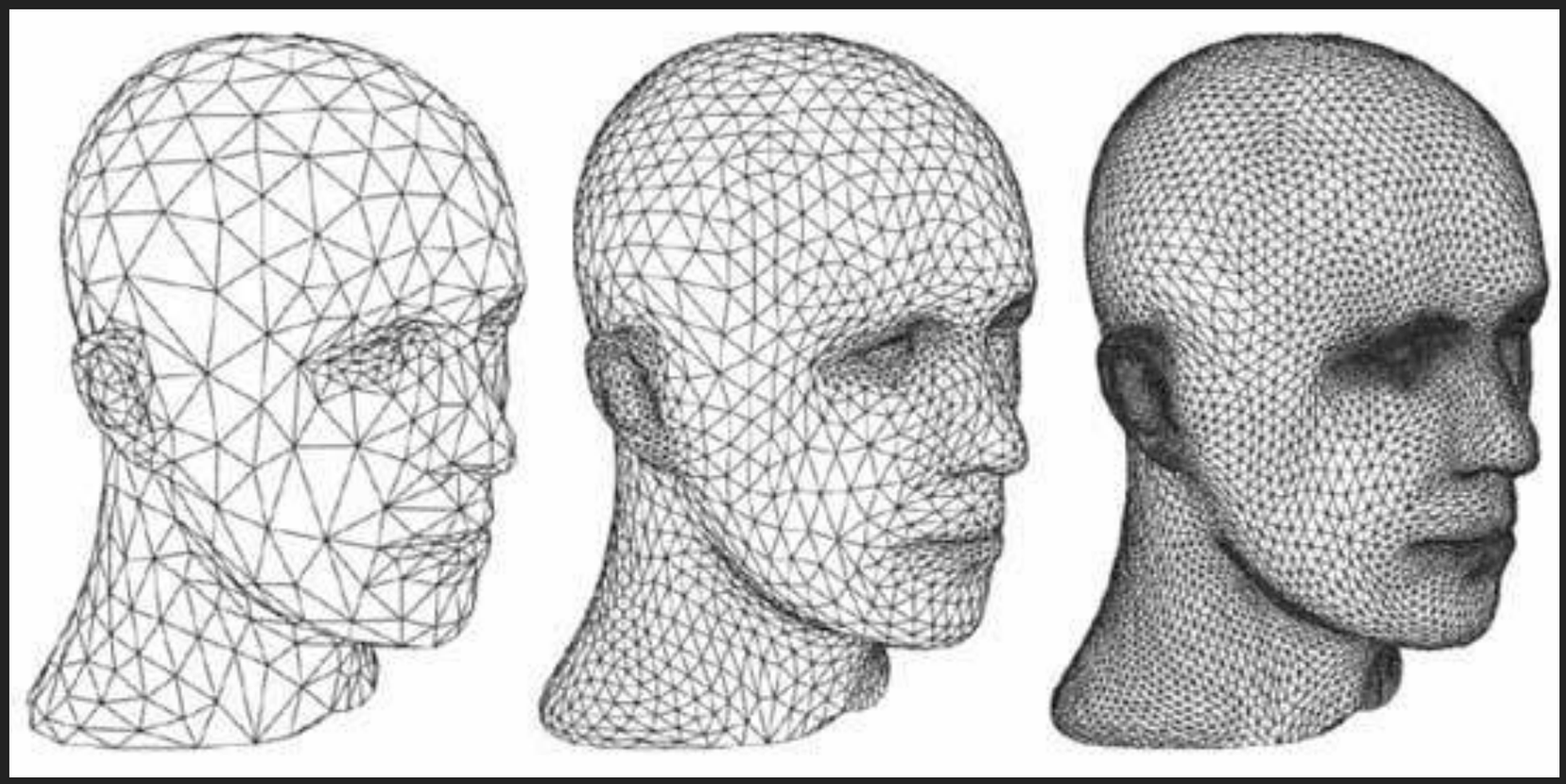
THE TRIANGLE



HTTP://WECLIPART.COM/TRIANGLE+SHAPES+CLIPART

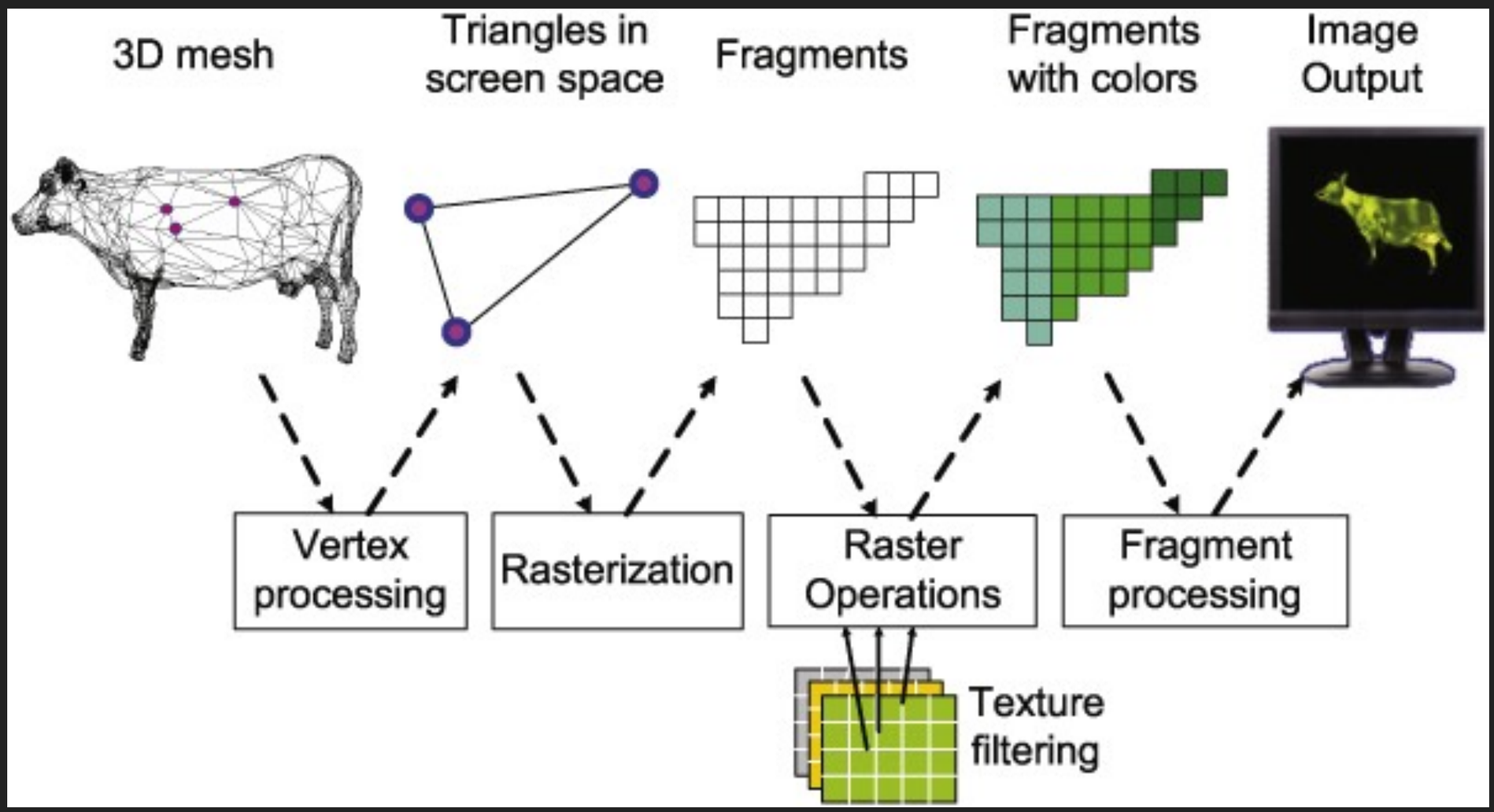
THE TRIANGLE

- Simplify the rendering problem:
- Export complex models into triangle meshes (offline process)
- Every frame:
  - Perform transformations on the model's vertices
  - Calculate color for each resulting pixel
- It's all adding, multiplying, texture lookups, trig and square roots



PROGRAMMABLE GRAPHICS PIPELINE

- The triangles are fed through a GPU (Graphics Processing Unit) pipeline
- A “Vertex Shader” transforms the vertices from objects space to screen space
- A “Fragment Shader” fills in the color



HTTP://IOPSCIENCE.IOP.ORG/ARTICLE/10.1088/1742-5468/2009/06/P06016/FULLTEXT/#JSTAT313542BIB14

VERTEX SHADER CODE

```
// Vertex Shader

// Constants
uniform mat4 modelMatrix;
uniform mat4 viewMatrix;
uniform mat4 projectionMatrix;

// Per-Vertex Data
attribute vec4 vertex;
attribute vec4 normal;
attribute vec2 texCoord;

// Output Data
varying vec3 vPosition;
varying vec3 vNormal;
varying vec2 vTexCoord;


void main() {
    // Compose the modelView Matrix
    mat4 modelViewMatrix = viewMatrix * modelMatrix;

    // Transform the input vectors
    vPosition = modelViewMatrix * vertex;
    vNormal   = normalize(modelViewMatrix * normal);
    gl_Position = projectionMatrix * vec4(vPosition, 1);

    // Texture coords are passed through
    vTexCoord = texCoord;
}
```

FRAGMENT SHADER CODE

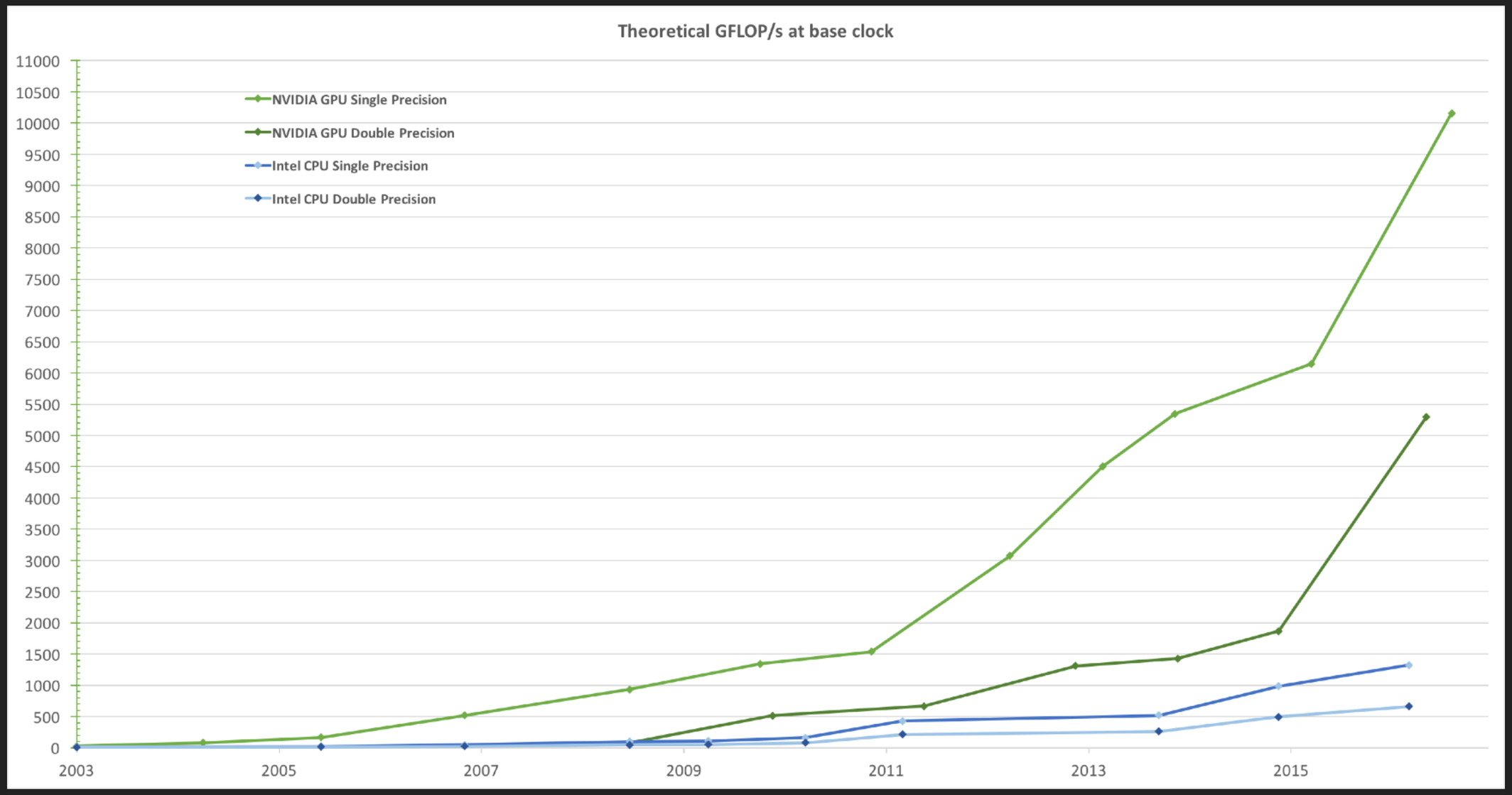
```
// Fragment Shader

// Light Constants
uniform vec4 lightPosition;
uniform vec4 lightColorDiffuse;
uniform vec4 lightColorAmbient;
uniform float lightDiffuse;

// Texture Map
uniform sampler2D texture;

// Per-pixel Interpolated Data
varying vec3 vPosition;
varying vec3 vNormal;
varying vec2 vTexCoord;


void main() {
    // Calculate diffuse light value
    vec3 L = normalize(lightPosition.xyz - vPosition);
    vec4 Idiff = lightDiffuse * max(dot(vNormal, L), 0.0);
    Idiff = clamp(Idiff, 0.0, 1.0);

    // Texture Lookup
    vec4 texColor = texture2D(texture, vTexCoord).bgra;

    // Final color value
    gl_FragColor = Idiff * lightColor * texColor;
}
```

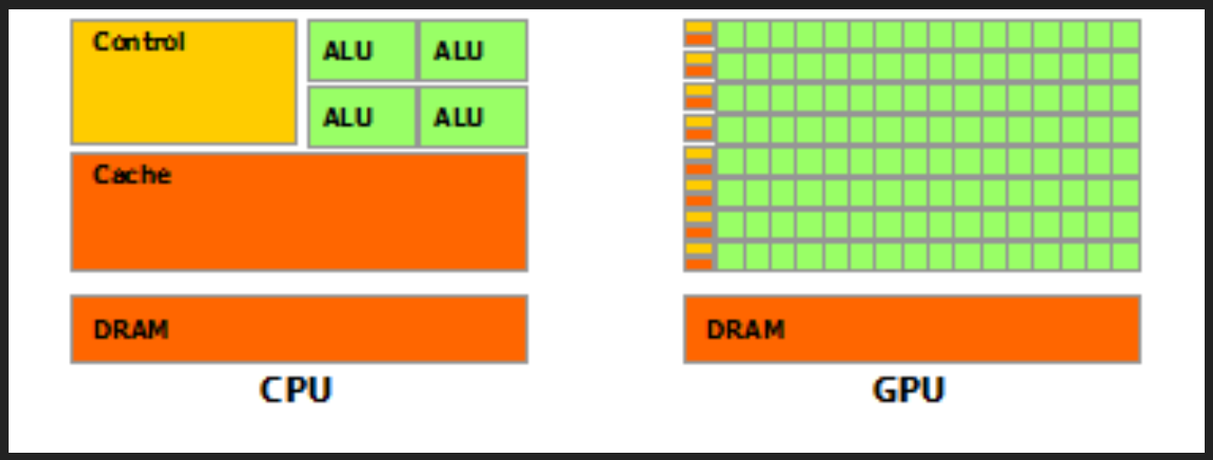
GPU – GRAPHICS PROCESSING UNIT

- CPU - Central Processing Unit
  - Intel, AMD. Handles very general computing needs.
  - Multi-core: often 2 or 4, up to 28; High Freq: 2 - 5 GHz
  - Some specialization added: math parallelism, crypto, compression

- GPU - Graphics Processing Unit
  - Nvidia, AMD (ATI). Very specific capabilities
  - Huge core count: 1000 to 5760 on latest! Lower Freq: 0.7-1.5 GHz
  - Monster at cranking vertices / fragments…
    - Adding, Multiplying, Texture Lookups (with very fast memory)
    - Specializations like trig functions

GPU – PERFORMANCE TRENDS





http://docs.nvidia.com/cuda/cuda-c-programming-guide/

GAMES – MORE PROGRAMMING!

- Terrain, Physics, Vehicles
- Animation, Skinning
- Cloth, Hair, Water
- Particles, Special Effects
- Heads Up Display and 2D overlay
- Artificial Intelligence, Path Planning
- Audio and Music
- Artists Tooling , Optimization and Testing
- Software Development Lifecycle Management

GAMES – NOT JUST FOR PROGRAMMERS

- Artists!
  - Sketching, Character Design, Costume Design
  - Modeling, Animation, Skinning, Texturing, Lighting
  - Architecture, Environments, Terrain
- Musicians and Audio Design
- Game Design, Level Design, Story, Copy Editing, QA
- Business, Accounting, Management, Sales and Marketing

BEYOND GRAPHICS

- GPUs are highly parallel multiply & adders
- Used for non-graphical problems
  (General Purpose GPU: GPGPU)
  - Physical Simulation (cloth and water in games)
  - Financial Modeling, Oil Exploration
  - Hashes for Cryptocurrency Mining
  - Machine Learning and Inference
    - Next up Tensor Processing Units… TPUs…

THANK YOU!

- Find the slides:
  - https://github.com/neomantra/presentations