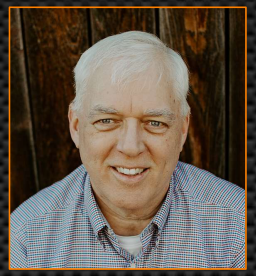
# GLSL: Origins, Observations, and Opportunities

Randi Rost
Randi Rost Consulting

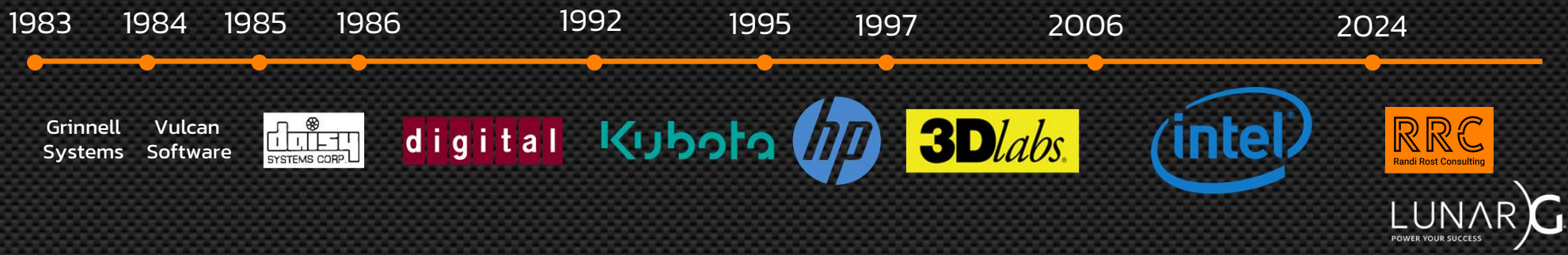


# Overview

- My background
- A few words about software
- A brief history of shading
- Shading language origins
- Creation of the OpenGL Shading Language
- Where are we headed?

# About me

- Software engineer
- Driven by:
  - Love for computer graphics
  - Desire to help developers succeed
- 15+ years of OpenGL



# When I was starting out...

- Early cloud computing!

Low speed networking

Manual network connections

Backup Storage

Non-graphical, low speed, text-only display

Non-ergonomic keyboard

Local storage

Non-adjustable work surface

# My first PC


# “It’s computer graphics!”

## Ken Perlin

*An Image Synthesizer*

Ken Perlin
*An Image Synthesizer*
SIGGRAPH ‘85 Proceedings
Vol. 19, No. 3, (July 1, 1985), 287–296
https://dl.acm.org/doi/10.1145/325165.325247

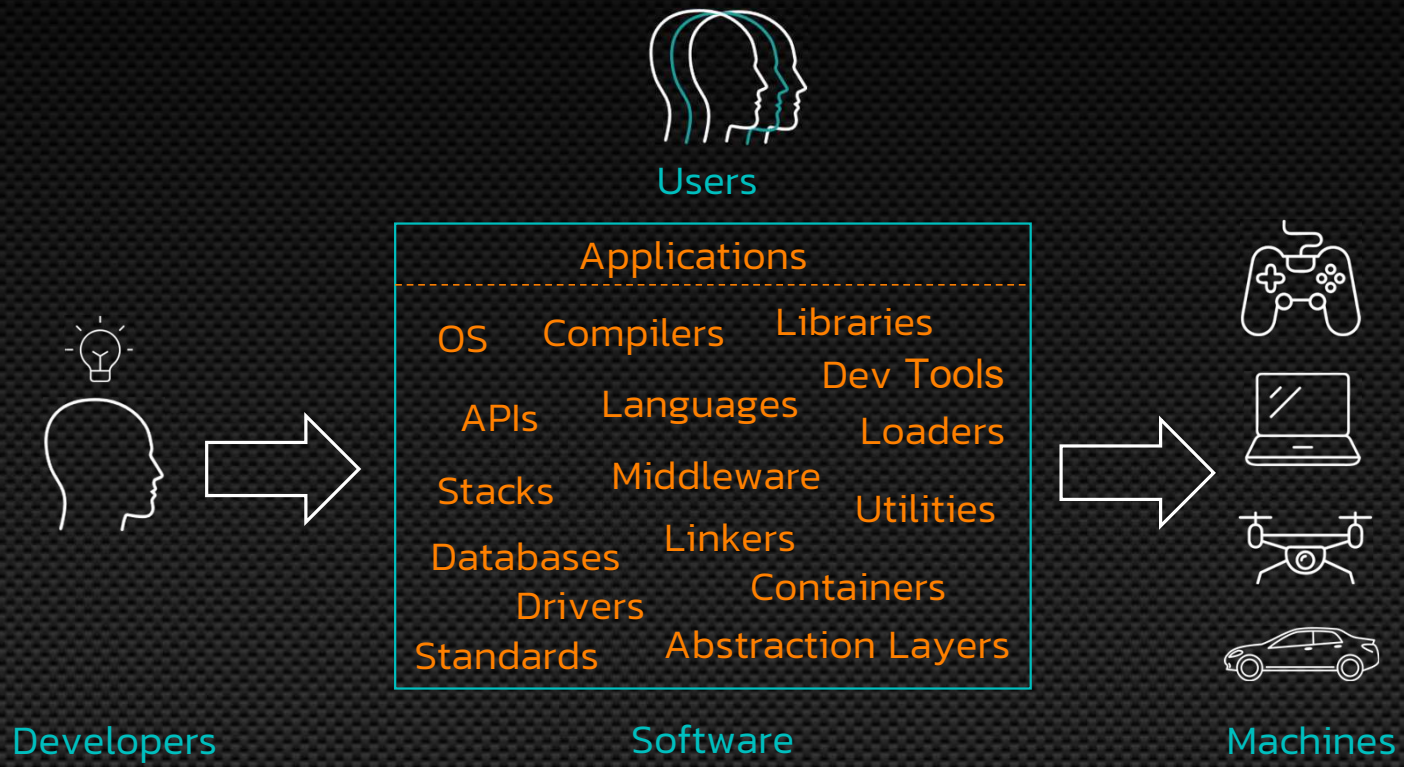
*Marble Urn*, © Ken Perlin. Reproduced with permission.

# Observations

## Software and software development

What is software, really?

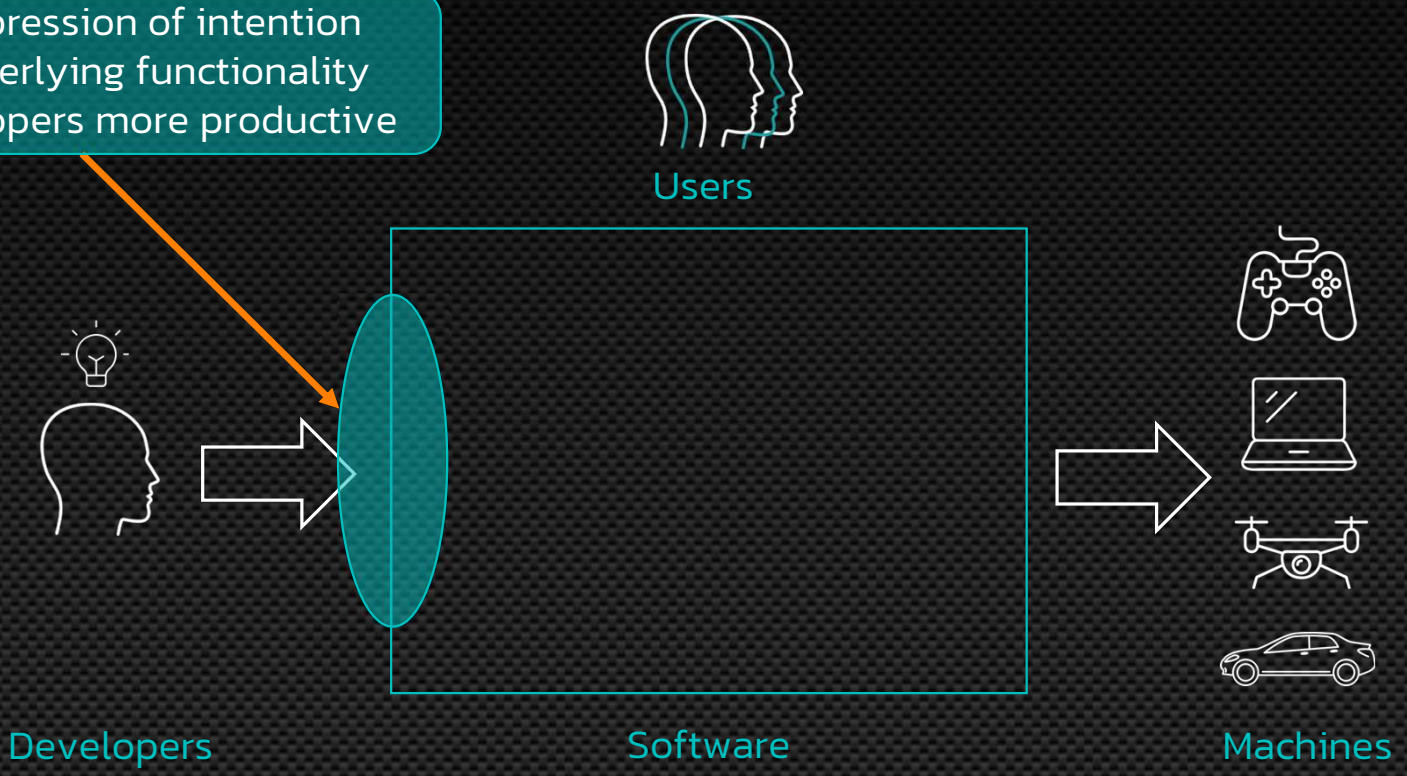
# What is software?




Software is the mechanism for turning ***human intention*** into ***machine execution***

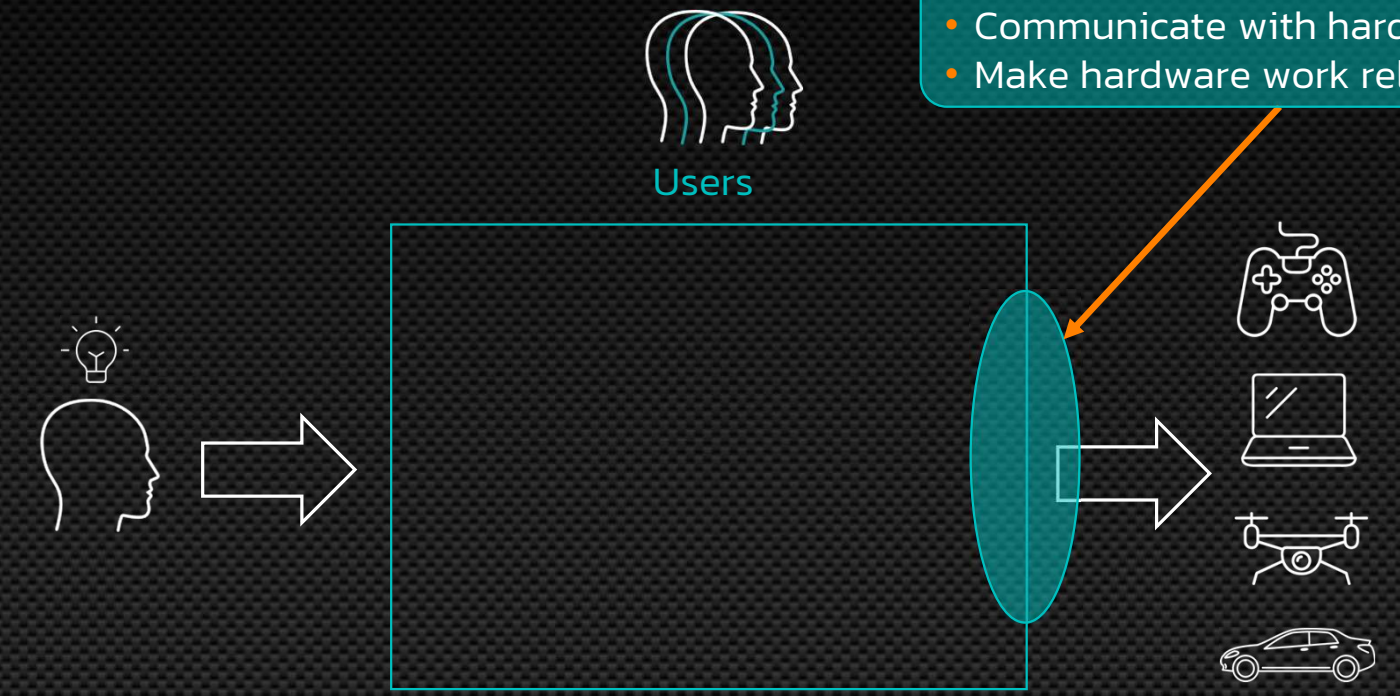
# Developer-facing software

- Simplify expression of intention
- Express underlying functionality
- Make developers more productive



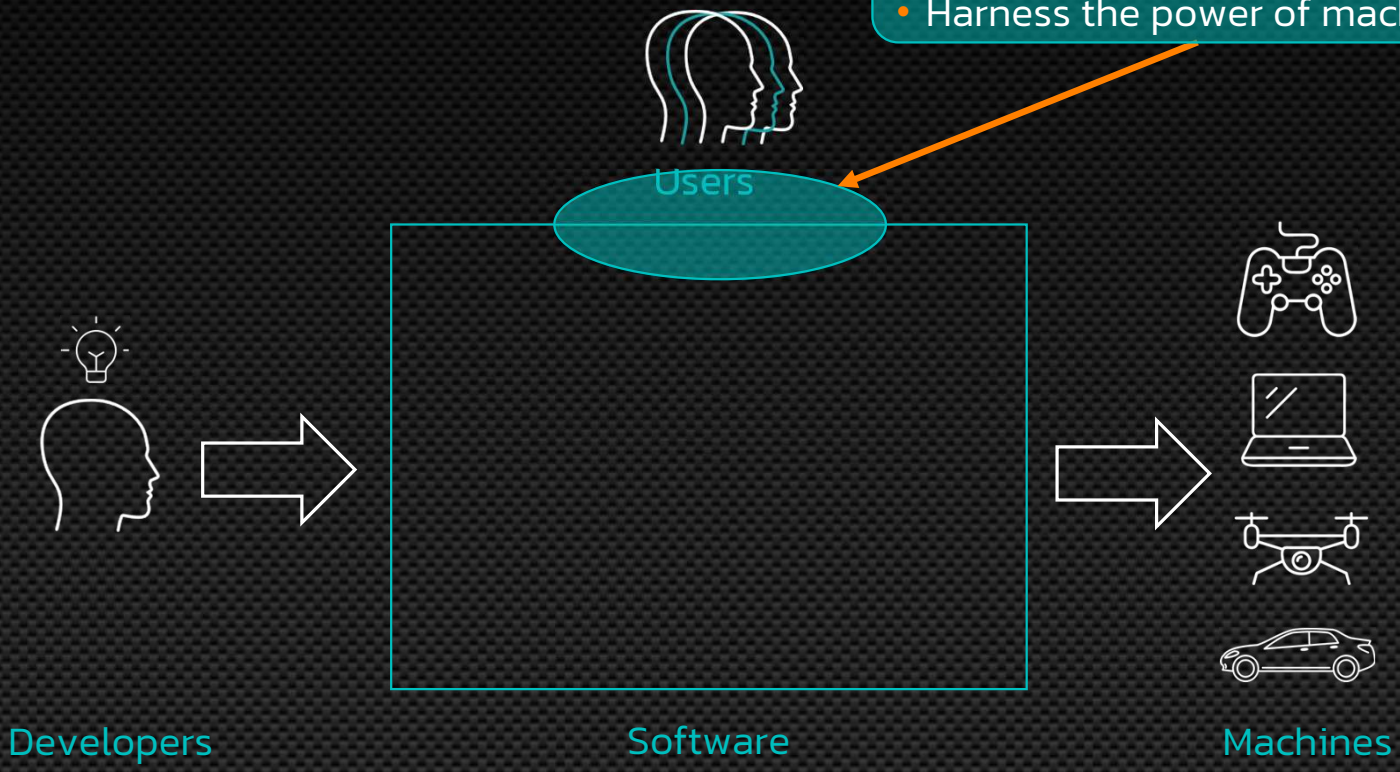
# Machine-facing software

- Control hardware functions
- Communicate with hardware
- Make hardware work reliably



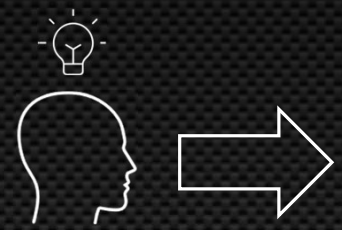
# Application software

- Improve productivity
- Unleash creativity
- Harness the power of machines



# How to tame software complexity

Developers



Software

Machines

Narrow the domain to a specific set of intentions

Determine the best way to translate those intentions into machine execution

Narrow the set of target devices

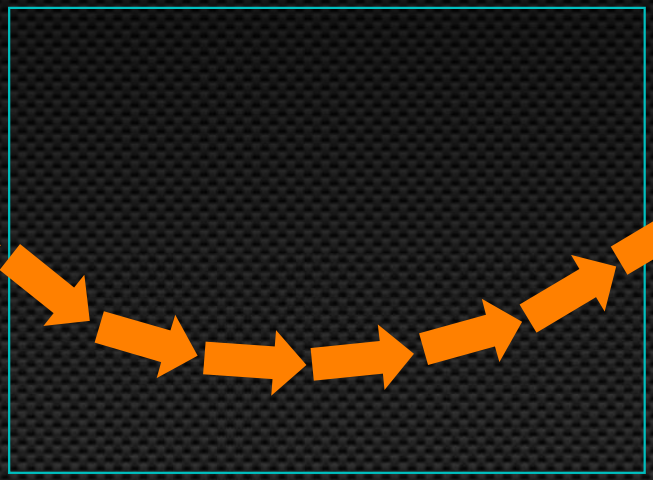
# The GLSL Vision

RRC
Randi Rost Consulting

Developers

Software

Machines



Interactive, 3D visualization

Industry standard, cross-platform, programming language for OpenGL

PCs with programmable GPUs

# Origins

## The path to interactive, realistic 3D

Shading techniques and shading languages

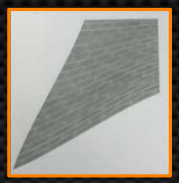
# A brief history of shading – 1970's

RRC
Randi Rost Consulting

1970 1971 1972 1973 1974 1975 1976 1977 1978 1979

Gouraud shading

Phong shading

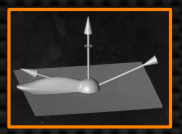


Texture mapping

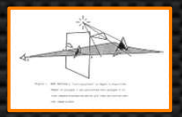
Fractals

Environment mapping

Antialiasing



Torrance-Sparrow reflection



Shadow volumes

Alpha channel

Shadow mapping

Bump mapping

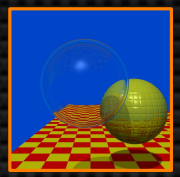
Transparency & Refraction

AtomLLL

# A brief history of shading – 1980's

RRC
Randi Rost Consulting

1980 1981 1982 1983 1984 1985 1986 1987 1988 1989

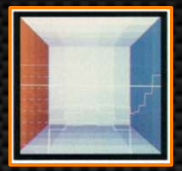


Ray tracing

Material-based reflection

Cloud/dust reflection

Particle systems



Radiosity

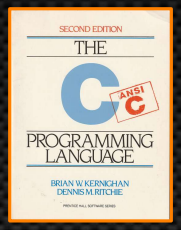
Motion blur
Depth of field
Soft shadows

Procedural textures

Path tracing
Caustics

# A brief history of shading languages

1970 1975 1980 1985 1990 1995 2000 2005 2010 2015

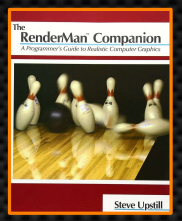



C

Software Testbed

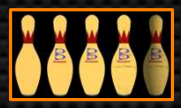
Image Synthesizer

Shade Trees




RenderMan

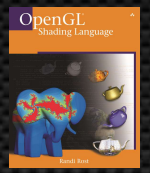
pfMan



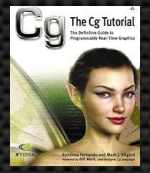
RTSL

OSL

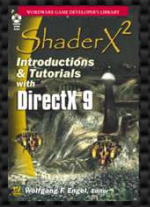
Metal Shading Language




GLSL




Cg




HLSL

"Of all of the things I've contributed to the field of computer graphics, I think that introducing what we now call procedural shaders was by far the most important." *Ken Perlin*

# Industry Status c. 2000

- OpenGL still compatible with 1.0 from 1992
- 230+ OpenGL extensions
- Graphics processors increasingly programmable via ASM-like interfaces
  - DirectX 8 vertex shader and pixel shaders
  - NVIDIA vertex programs, fragment programs, register combiners, texture shaders
  - ATI vertex shaders, fragment shaders
- System architecture significantly different from 1992
- Microsoft innovating rapidly with Direct3D

Opportunity for 3Dlabs to make a bold proposal – OpenGL 2.0

# What was "OpenGL 2.0"



- A vision developed by extensive ISV feedback that proposed a variety of new features for OpenGL
  - A high-level shading language
  - Unified model for objects
  - More user control over memory management
  - Better time management facilities (driven by OpenML)
  - Support for asynchronous operations
  - Configurable frame buffer and offscreen rendering
  - Elimination of many extensions

# GLSL Goals

- Define a language that is specific to OpenGL ✓
  - (Eventually) replace fixed function with programmability
  - Eliminate the need for numerous extensions
  - Add API extensions to create, manage, and execute shaders
  - Provide access to existing state
- Expose the flexibility of present and near-future hardware ✓
- Provide hardware independence ✓
- Define a language that is easy to use ✓
- Define a language that will stand the test of time ✓
  - Still useful in 10 years

# Shading language milestones

RRC
Randi Rost Consulting

3Dlabs

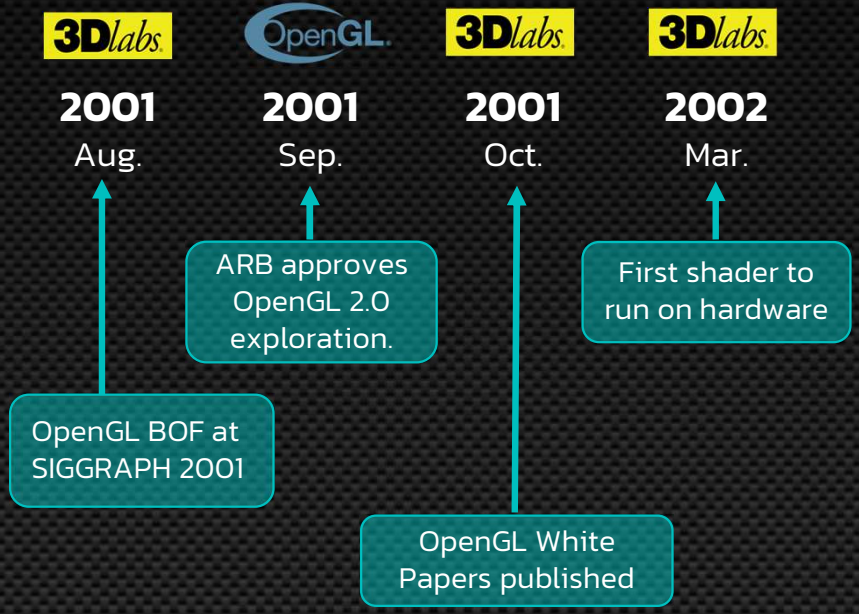
**2001**
Aug.

OpenGL BOF at SIGGRAPH 2001

# ARB meeting to propose OpenGL 2.0


*9/11 World Trade Center Attack*, 2001. Creator: Sean Adair. Credit: Reuters. Unmodified. Creative Commons License.

# Shading language milestones



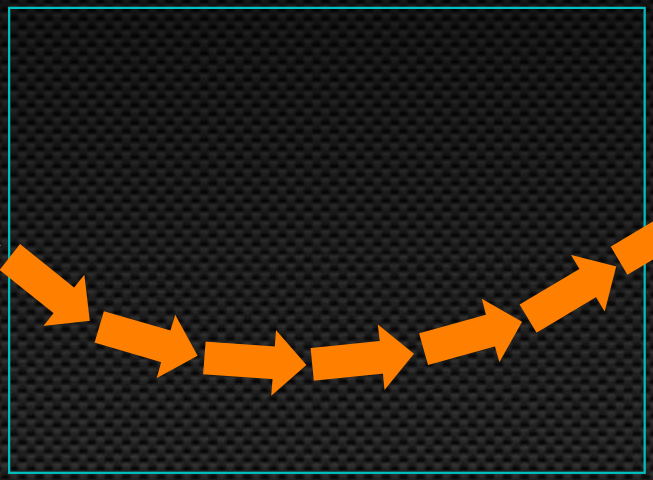
# The GLSL Vision

RRC
Randi Rost Consulting

Developers

Software

Machines

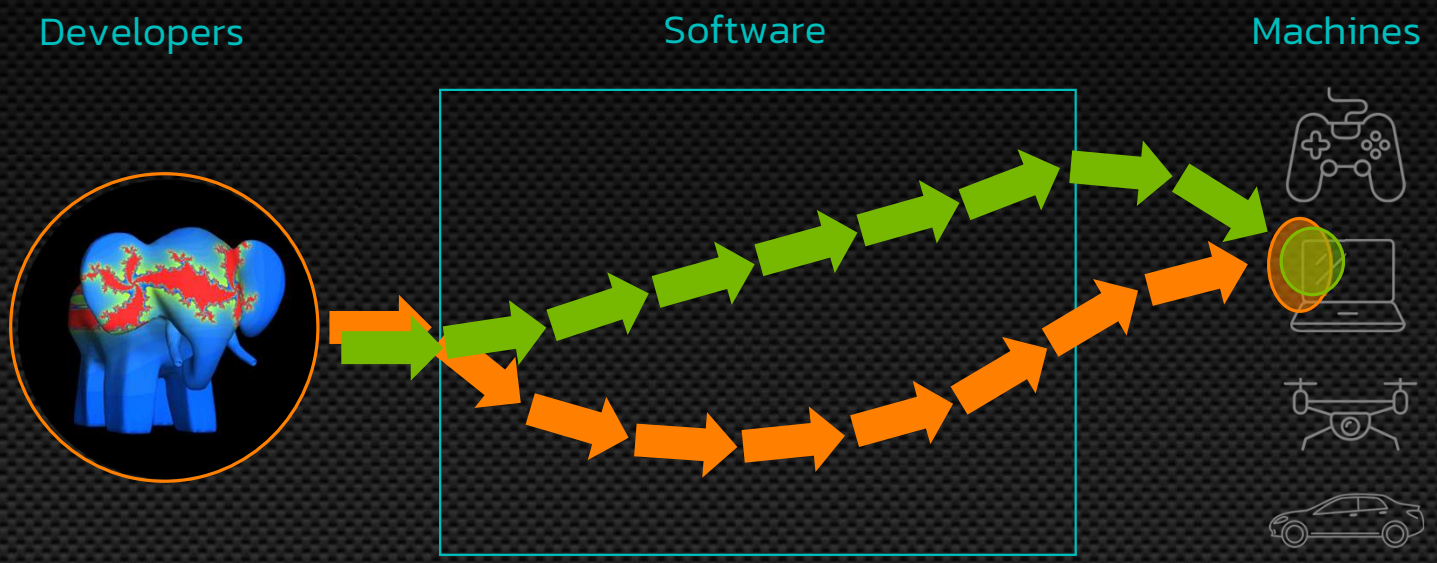


Interactive, 3D visualization

Industry standard, cross-platform, programming language for OpenGL

PCs with programmable GPUs

# Possible GPU vendor vision




Interactive, 3D visualization

Industry standard, cross-platform, programming language

Proprietary, GPU-specific programming language

PCs with programmable GPUs

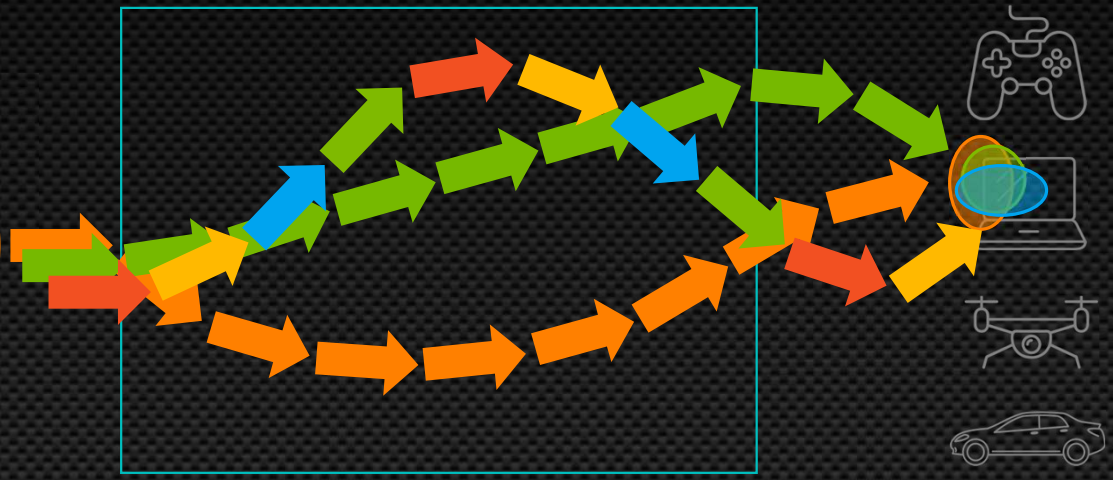
# Possible OS-vendor vision

RRC
Randi Rost Consulting

Developers

Software

Machines



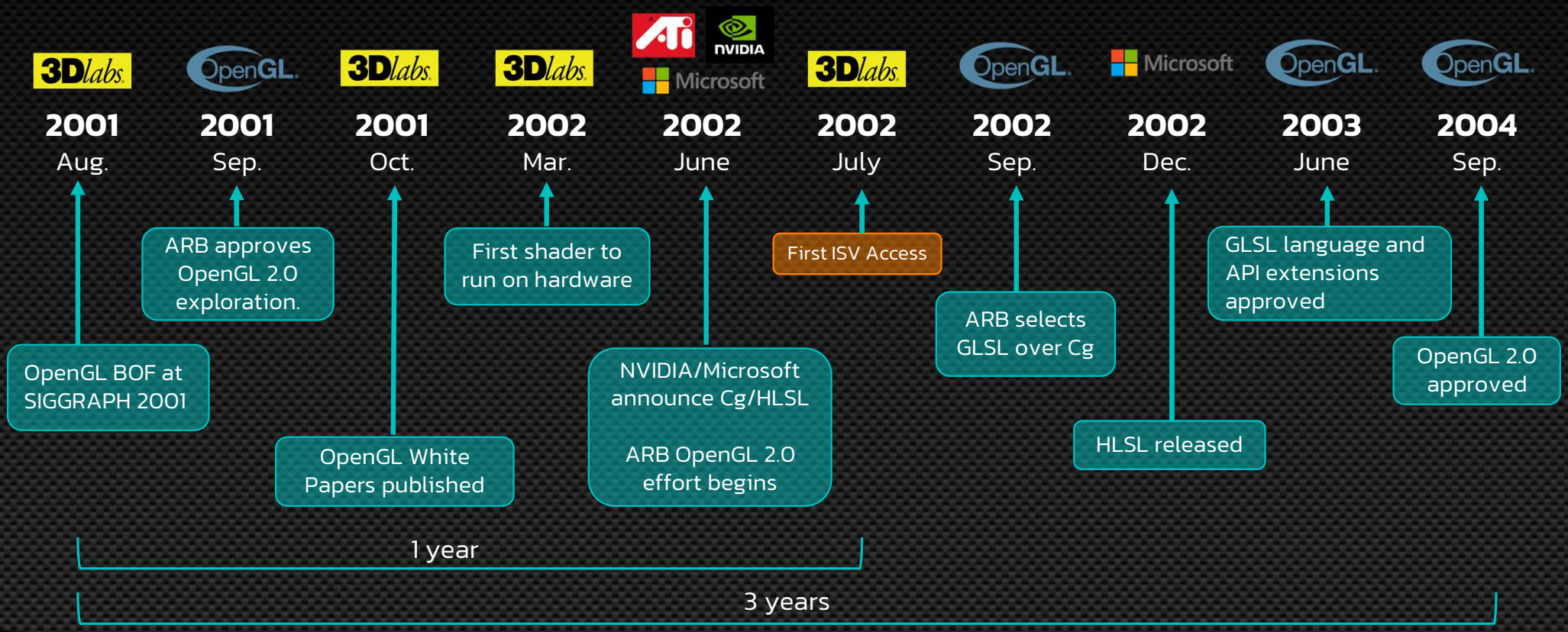


Interactive, 3D visualization

- Industry standard, cross-platform, programming language
- Proprietary, GPU-specific programming language
- Proprietary, OS-specific programming language

PCs with programmable GPUs

# Shading language milestones



# Proprietary vs. Standard Interfaces

RRC
Randi Rost Consulting

| Aspect | Proprietary | Standard |
|---|---|---|
| Cross-platform support | | ✓ |
| Cross-vendor support | | ✓ |
| Long-term code portability | | ✓ |
| Driver quality and bugs | ✓ | |
| Feature availability time-to-market | ✓ | |
| Performance | ✓ | |
| Learning curve/learning resources | ✓ | |
| Tooling and debugging | ? | ? |
| Ecosystem and middleware | | ✓ |
| Deployment and validation | ✓ | |
| Future-proofing | | ✓ |

# GLSL vs. Cg vs. HLSL

RRC
Randi Rost Consulting

| Aspect | GLSL | Cg | HLSL |
|---|---|---|---|
| Primary API | OpenGL | OpenGL/DirectX | DirectX |
| Syntax base | C-like | C-like | C-like |
| Vector types | vec{2,3,4} | float{2,3,4} | float{2,3,4} |
| Matrix types | mat{2,3,4}<br>Column major | float{2x2, 3x3, ...}<br>Row major | float{2x2, 3x3, ...}<br>Row major |
| Matrix multiplication | Mathematical | Component-wise | Component-wise |
| Compilation model | Run-time compilation by OpenGL driver | Offline or runtime via Cg compiler library | Compiled by DirectX, bytecode to driver |
| Cross-platform | OpenGL-level portability | OpenGL or DirectX | Windows/DirectX-native |

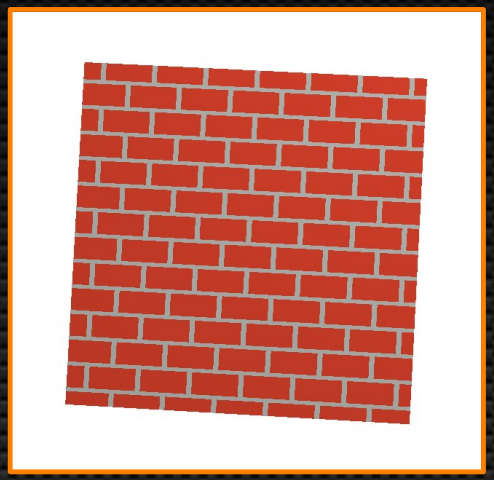
# Where are they now?

RRC
Randi Rost Consulting

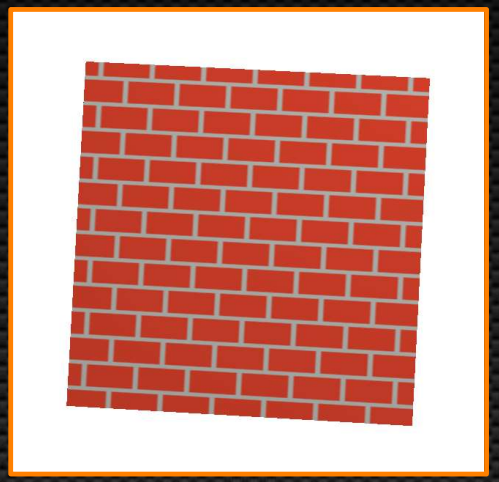
| | Market Cap |
|---|---|
| 3Dlabs | $0 |
| Microsoft | $3,460,000,000,000 |
| NVIDIA | $4,560,000,000,000 |

# The first GLSL shader


*Screen shot of output from brick shader.* March 2002.

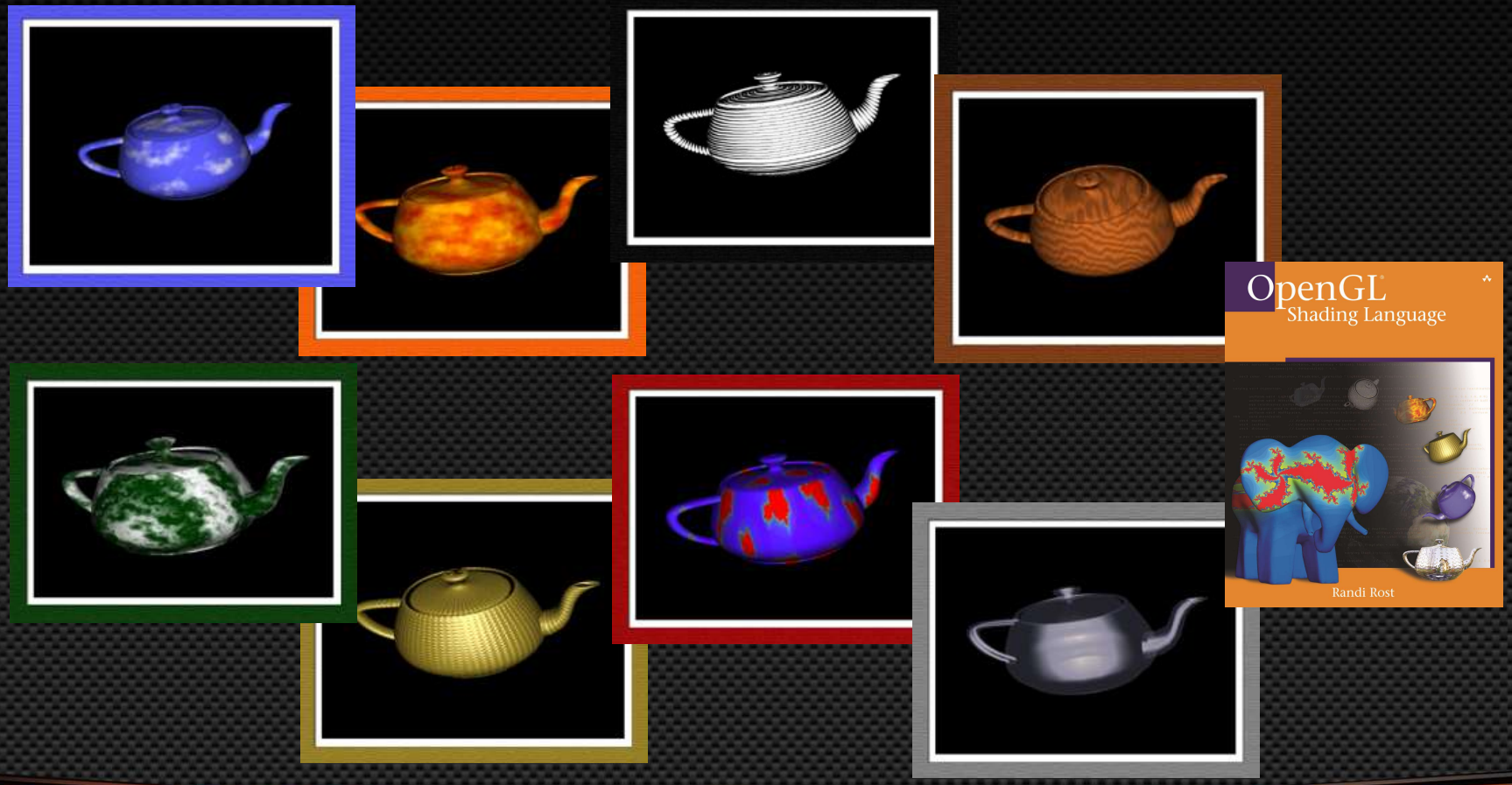


*Screen shot of output from aliased brick shader from the Orange book.*



*Screen shot of output from antialiased brick shader from the Orange book.*

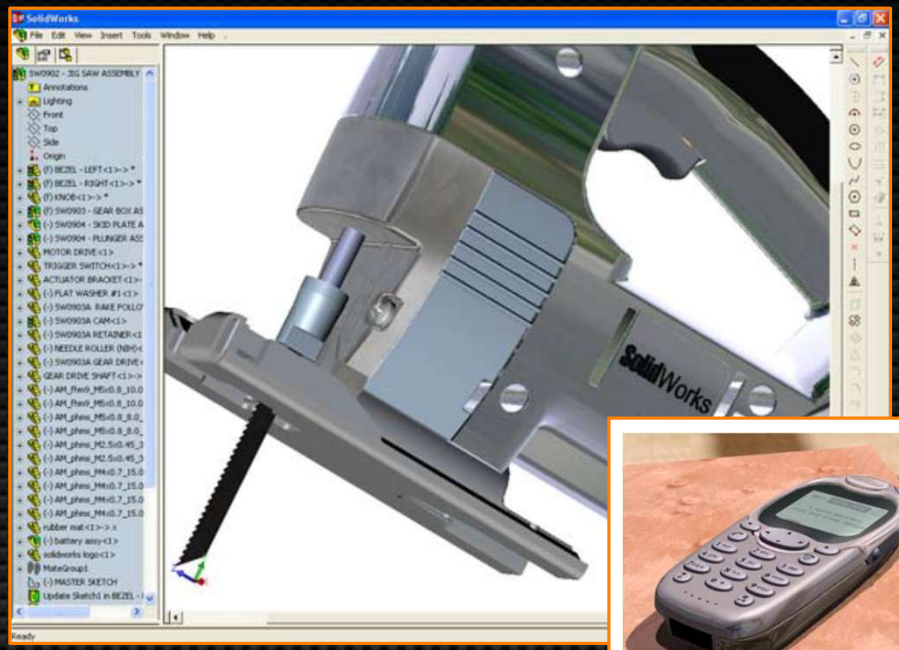
# Dawn of real-time programmability



# Other fun shaders

# Early ISVs

- SideFX Software
- Lightwork Design
- Pandromeda
- Alias
- Discreet
- Solidworks
- Adobe

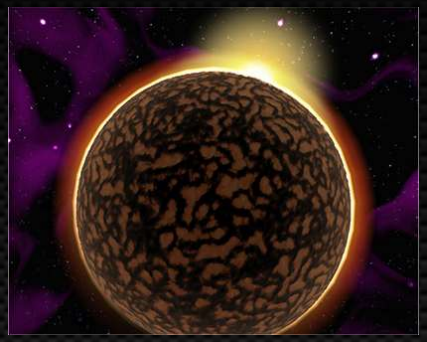


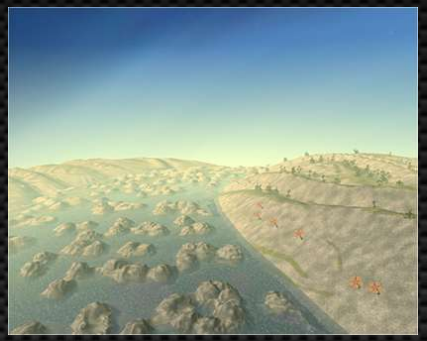
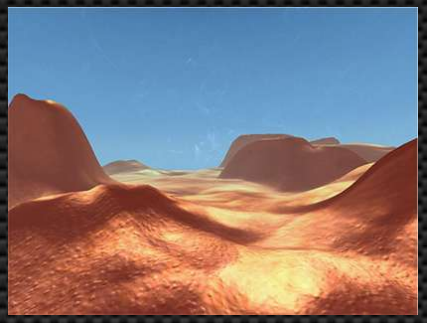
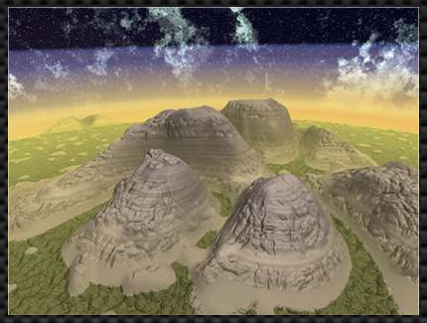

*Screen shot of GLSL rendering within Solidworks.* From the *OpenGL Shading Language, 2nd. Ed.* 2006

*Screen shot of GLSL rendering within Lightworks.* From the *OpenGL Shading Language, 2nd. Ed.* 2006

# RealWorldz (2005)

- Commissioned demo (Pandromeda)
- Everything generated procedurally
- Planets are mathematical spheres
- Interactive rendering
- Adjustable parameters for:
  - Cloud layers
  - Condensing/evaporating clouds
  - Cloud light diffusion
  - Ocean levels
  - Atmospheric density
  - Accurate sun halos
  - Particles for perspective haze
  - Darkening/flattening landscape contrast
  - Light scattering
  - Caustic reflections
  - Plants with growth cycles
  - Navigability anywhere in the world

# 3Dlabs implementation team

Shading Language Design
- **Dave Baldwin**
- John Kessenich
- Randi Rost

OpenGL API Design:
- Barthold Lichtenbelt
- John Kessenich
- Randi Rost

Standardization:
- Randi Rost
- Barthold Lichtenbelt
- John Kessenich
- Neil Trevett

Compiler Implementation:
- John Kessenich
- Greg Fischer
- Steve Koren

Driver Development:
- Barthold Lichtenbelt
- Matthew Williams
- Steve Koren
- Dave Houlton
- Russ Huonder
- Teri Morrison
- Na Li

Tools and Demos:
- Barthold Lichtenbelt
- Phillip Rideout
- Mike Weiblen
- Dave Houlton
- Joshua Doss
- Inderaj Bains
- Clifton Robin

**Please take a bow!**

# Other notable contributions

- Bill Licea-Kane (ATI) – GL 2.0 workgroup chair, early shader contributions
- Hugh Malan (Pandromeda) – RealWorldz developer
- Marc Olano (UMBC)
- ATI – Evan Hart, Jeremy Sandmel, Benjamin Lipchak, Glenn Ortner, Natasha Taturchuk
- NVIDIA – Steve Glanville, Cass Everitt, Pat Brown, Kurt Akeley
- SGI – Jon Leech
- Vital Images – Karel Zuiderveld, Matt Cruikshank, Steve Demlow
- Spinor – Folker Schamel
- Individuals – Allen Akin, Matt Spinor

**Please take a bow!**

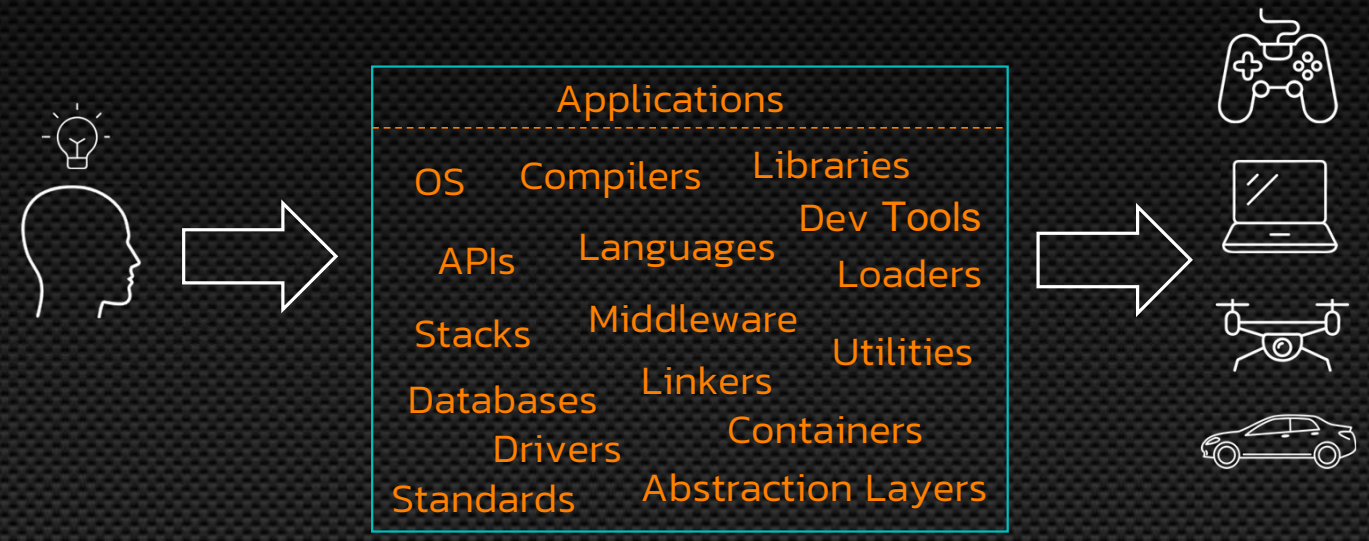
RRC
Randi Rost Consulting

# Opportunities

## Now what?

Maybe we should reinvent software?

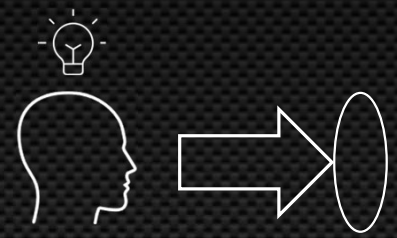
# What is the future of software?




If software is the mechanism for turning *human intention* into *machine execution*, is this what we'd invent if we were starting from scratch today?

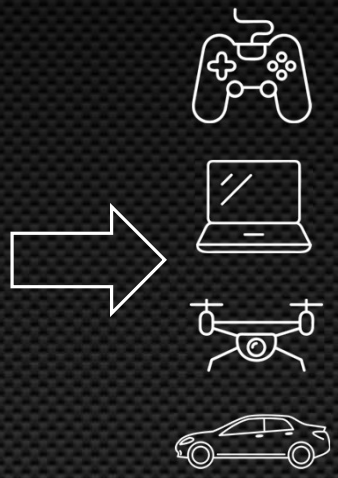
# What makes software difficult?

Developers



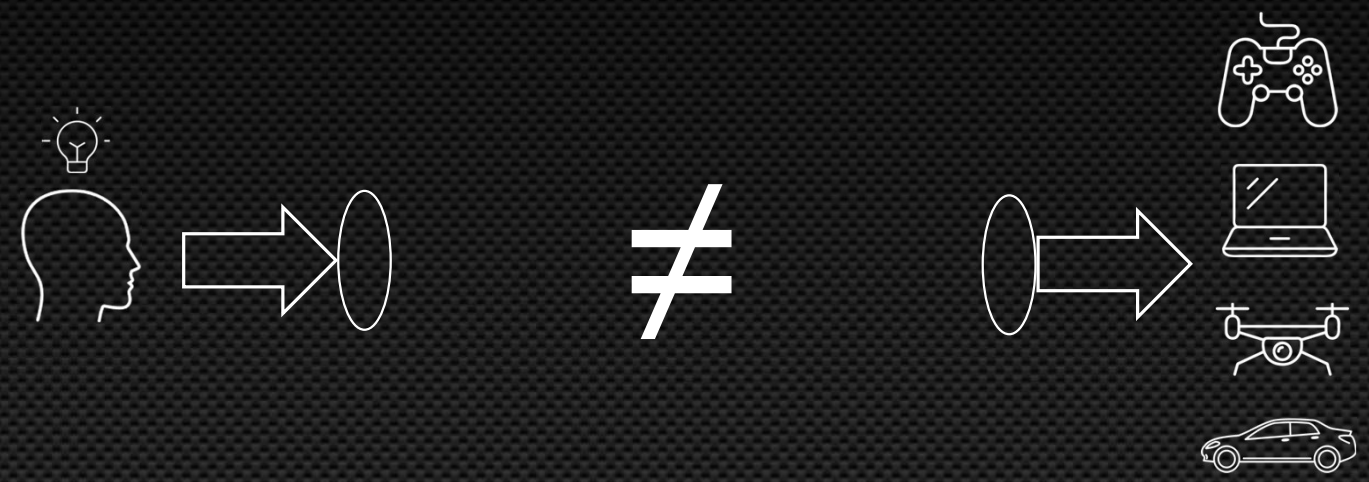
Software

Machines



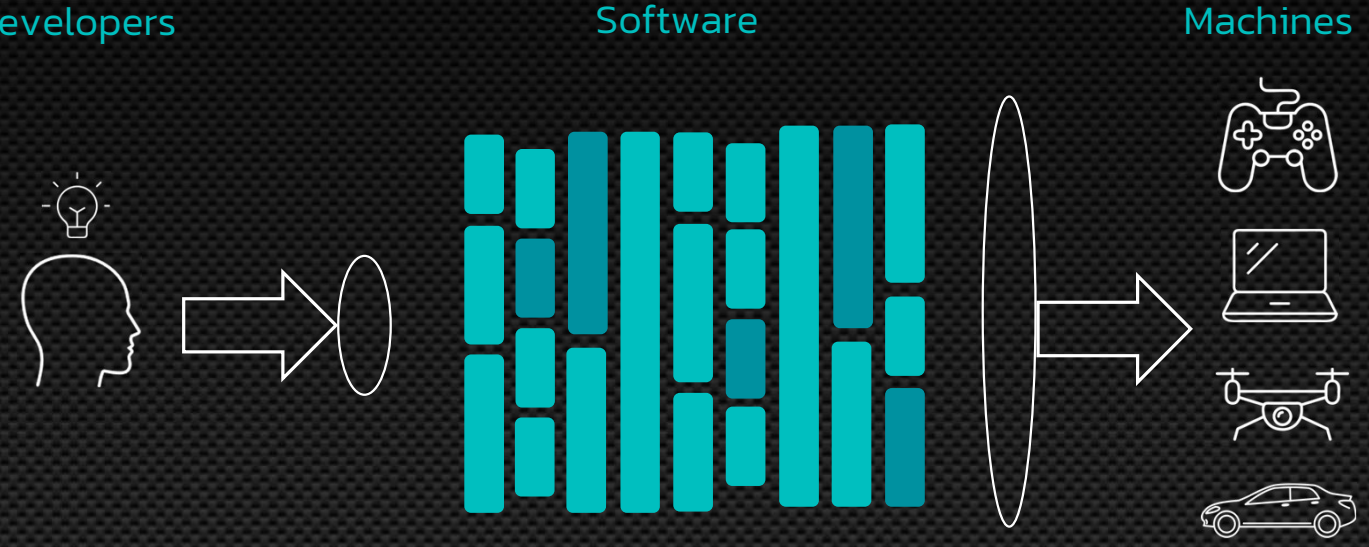
Humans have great difficulty expressing ourselves.

# What makes software difficult?



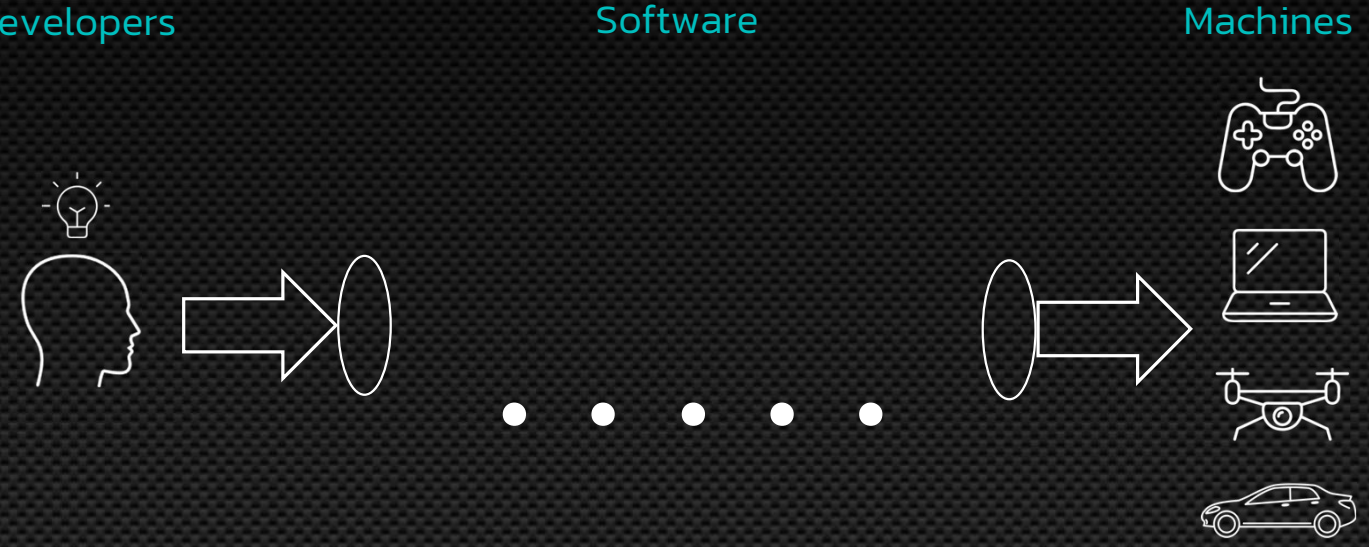

Human language is very different from machine language.

# What makes software difficult?




Software is so complex that we must rely on numerous layers to construct a large system. Layers don't eliminate complexity but hide it imperfectly.

# What makes software difficult?



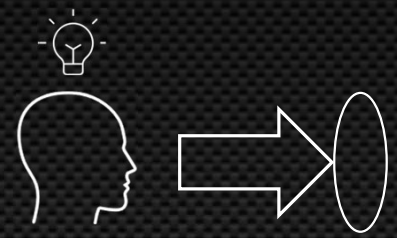

It can take years for new hardware to be properly supported by all the software that matters

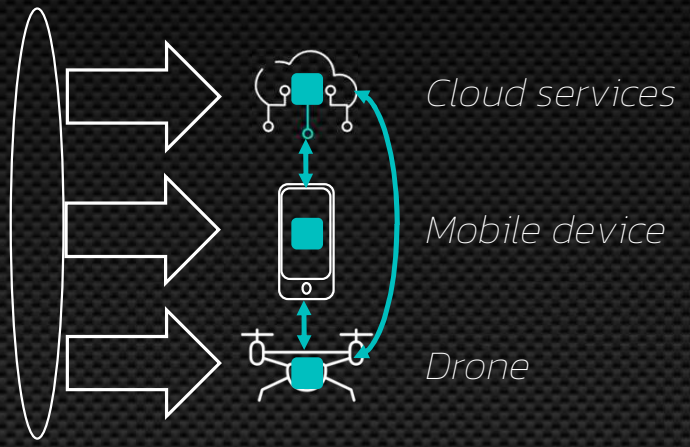
# What makes software difficult?

Developers

Software

Machines

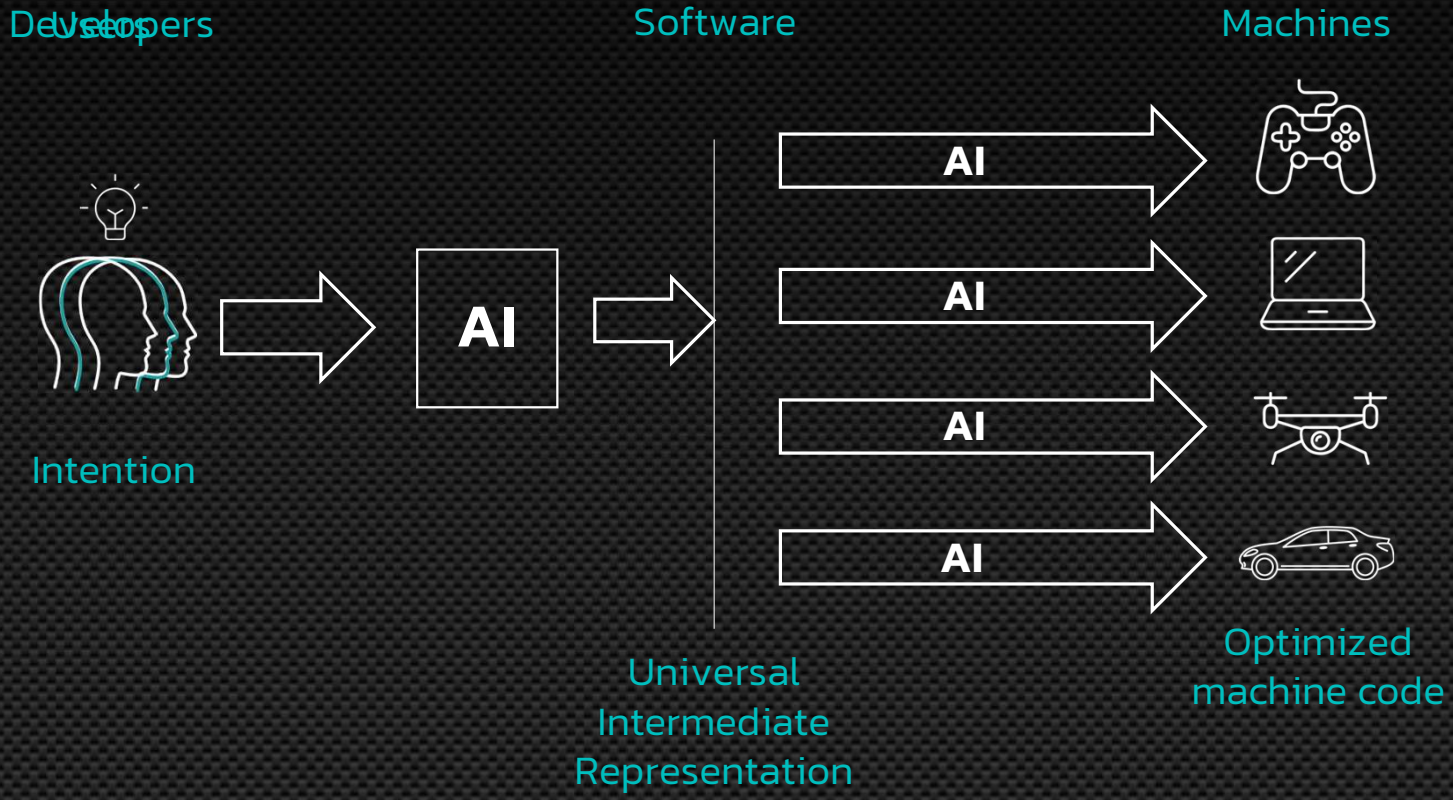




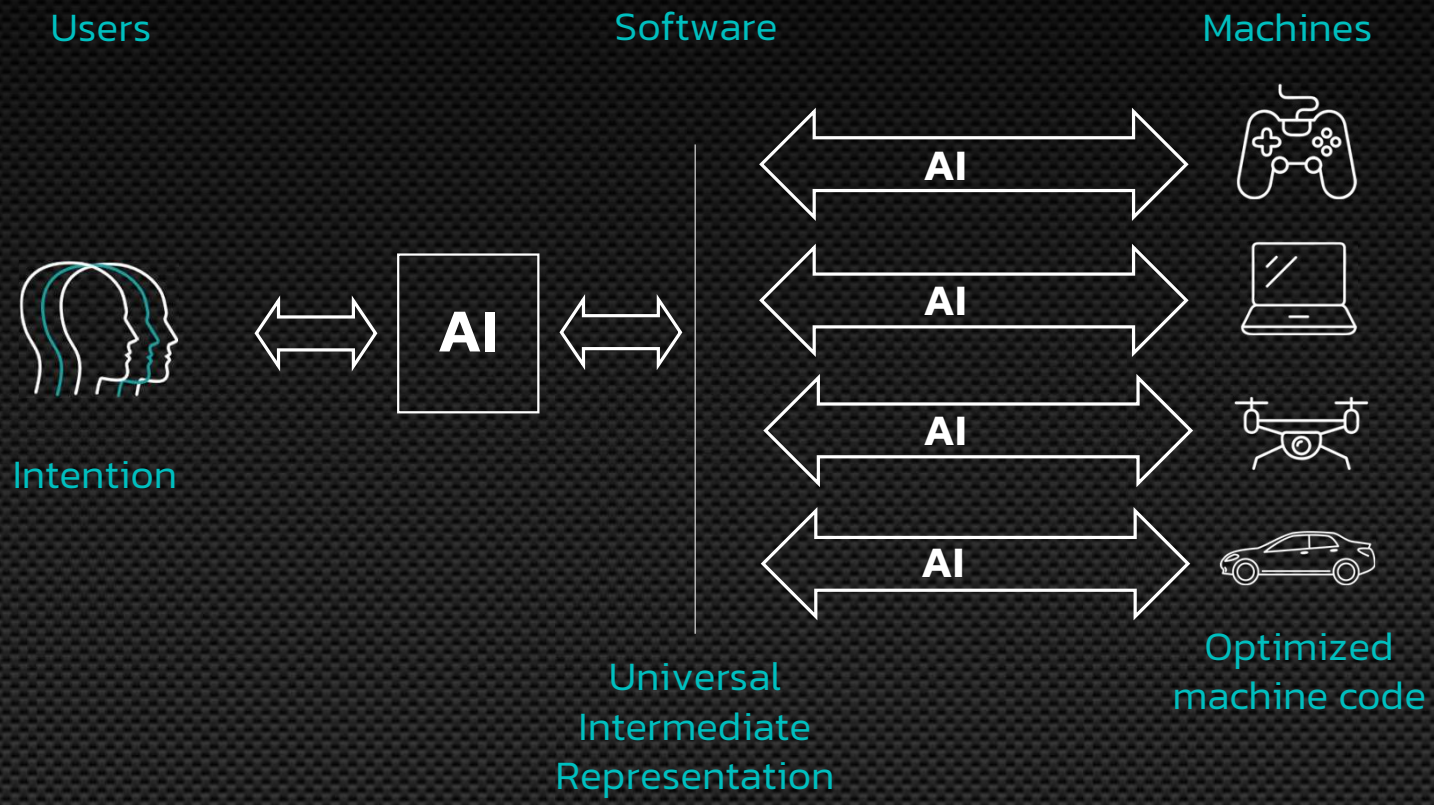

Modern applications often require code running asynchronously on distributed heterogeneous devices.

# An alternative future?

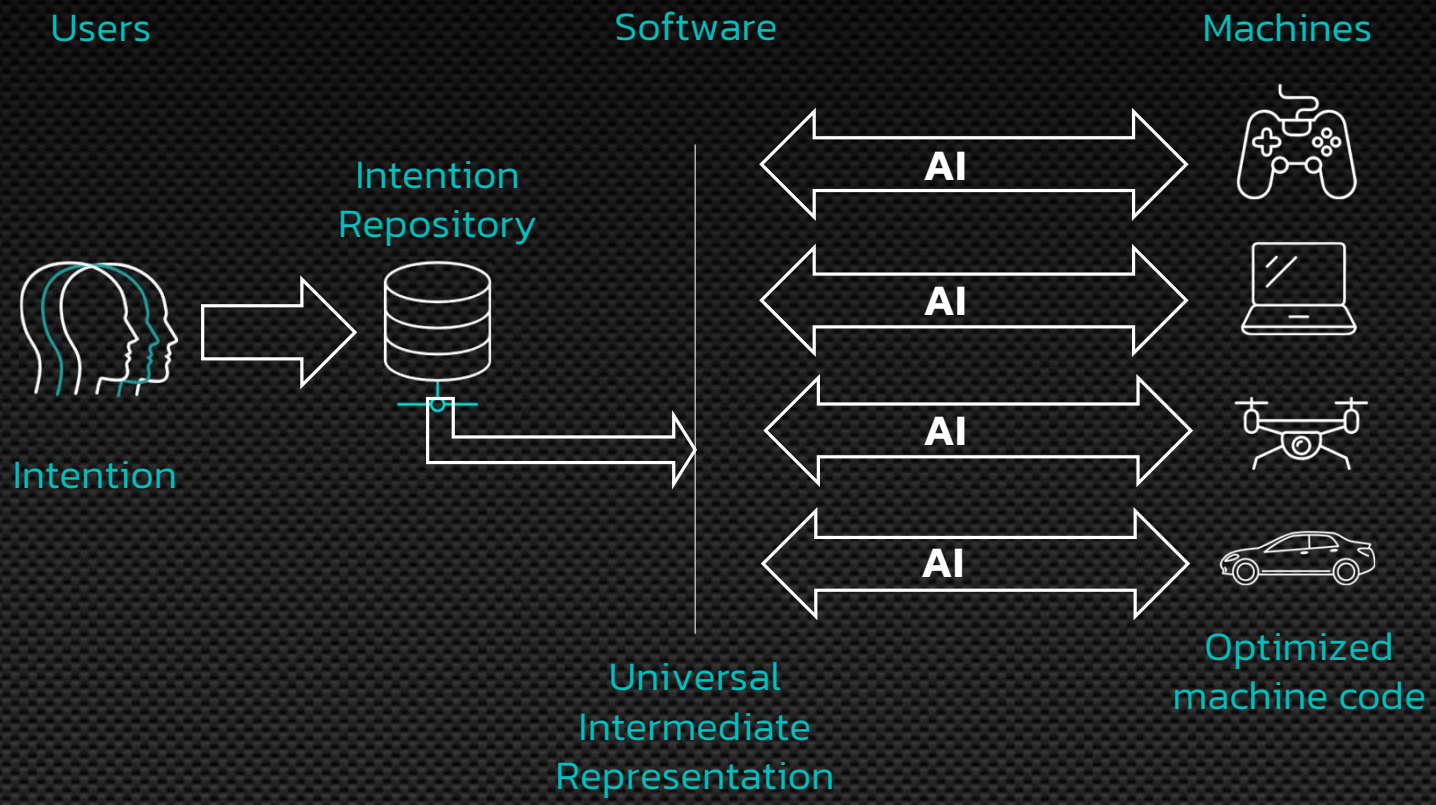
# Solving the software complexity problem



# Solving the under-specification problem



# Reusing intentions



# Benefits

- Everyone can create software
- Frequently used "intentions" get codified and can be accessed and reused
- Intent is completely separate from implementation
- "Code" has no bugs and is fully optimized for any hardware
- No hardware vendor lock-in – execute code instantly on any hardware
- New hardware gets immediate access to all previously written software

# Summary

- We've come a long way, baby!
- We owe thanks to others who paved the way for us and for the team that made GLSL happen.
- GLSL made shading:
  - So much faster
  - So much easier
  - For lots more people
- Maybe it's time to really focus on turning human intention into machine execution!

"It's one of those jaw dropping "wow" moments actually, so we thank you for making that happen! . . . It rocks. Having read the original white paper still did not prepare us to see it actually working. The ease with which we can now define & adjust OGL2 shaders is astonishing."
*Paul Salvini, CTO, Side Effects Software, July 2002*

RRC
Randi Rost Consulting
LUNARG

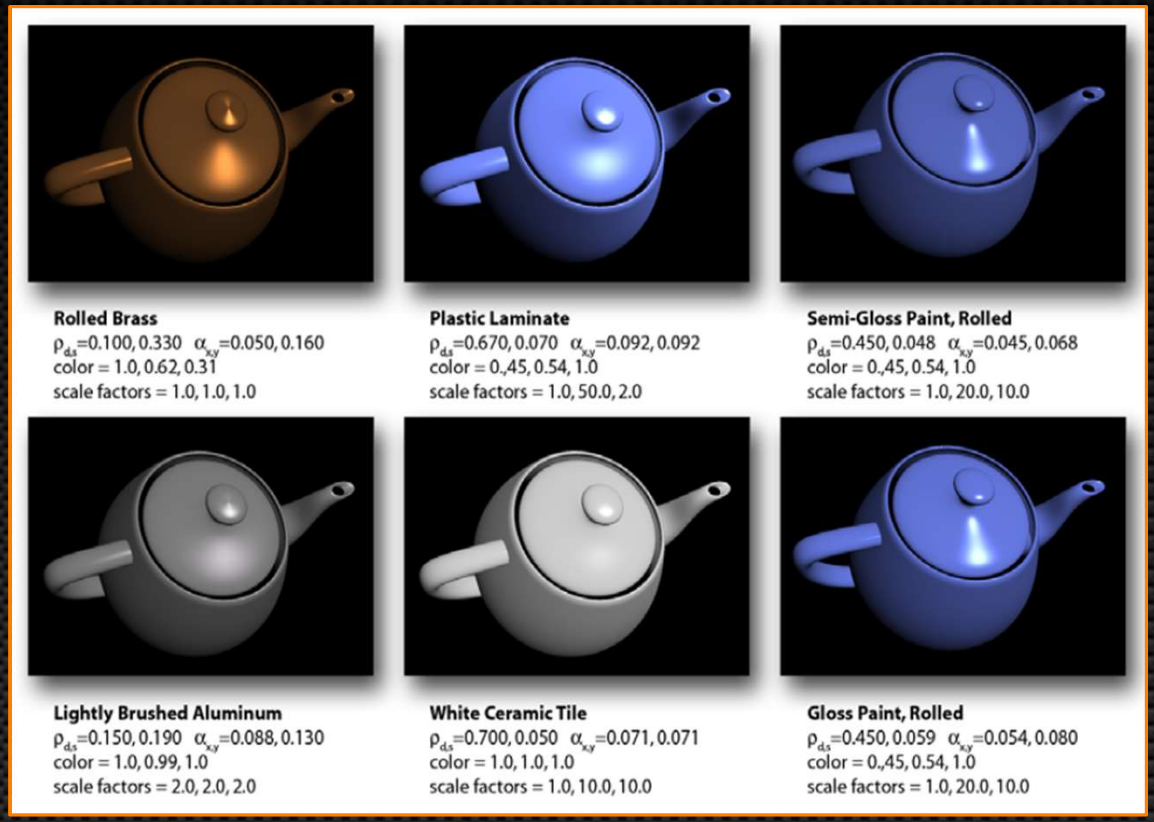
# Thank You!

# Backup

## Good, but not good enough

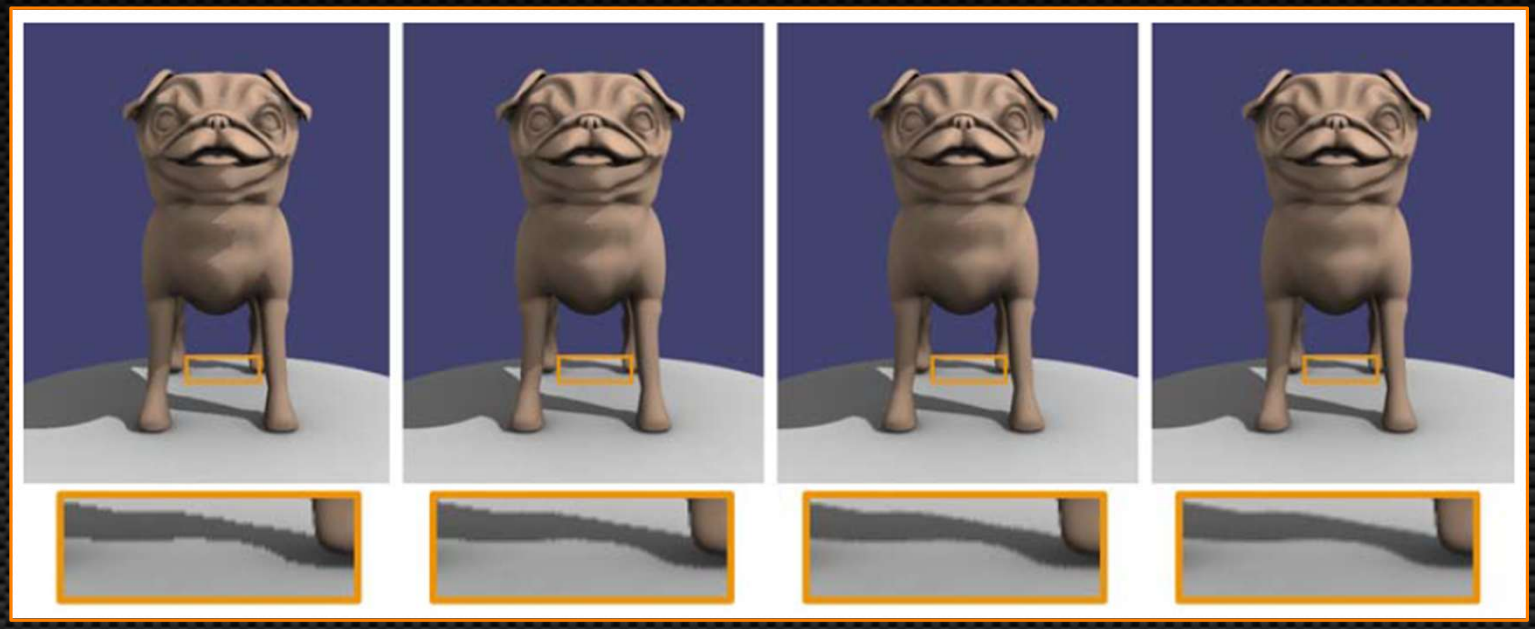
Or extra

# BRDFs




*Materials rendered with Ward's BRDF model and his measured/fitted material parameters.* From *OpenGL Shading Language, 2nd. Ed.* 2006

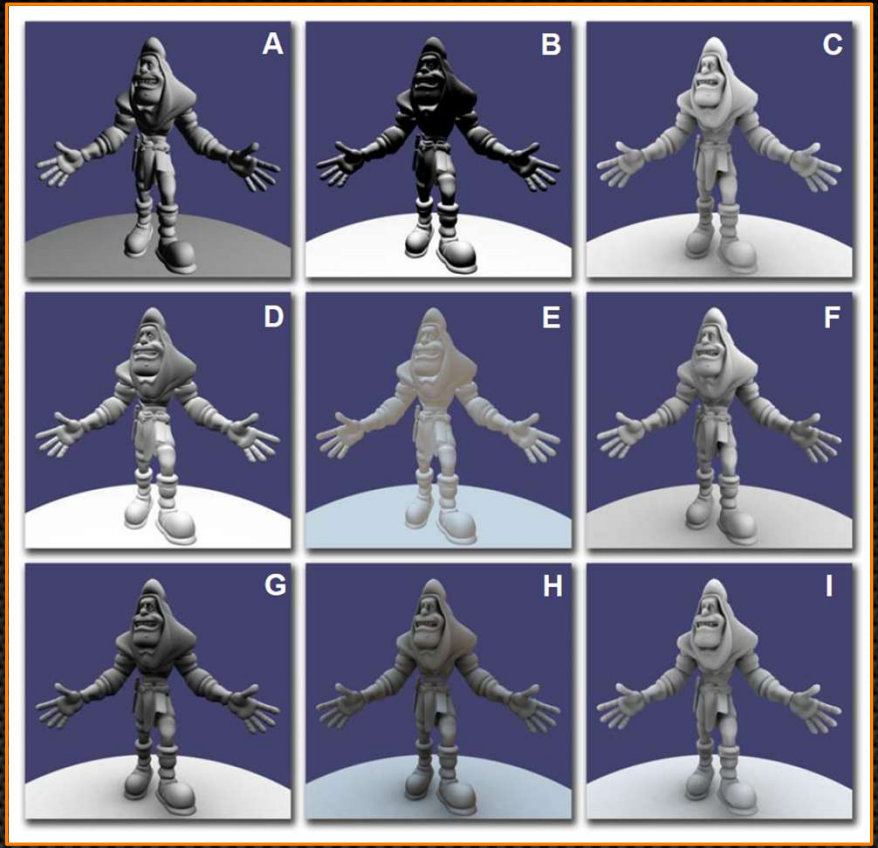
# Shadows



*A comparison of shadow mapping with one, 4, 4 dithered, and 16 samples per pixel.* From the *OpenGL Shading Language, 2nd. Ed.* 2006

# So much more flexibility!

A. Fixed functionality
B. Fixed functionality
C. Ambient occlusion
D. Hemisphere lighting
E. Spherical harmonics
F. Diffuse attenuated by ambient occlusion
G. Hemisphere lighting + ambient occlusion
H. Spherical harmonics + ambient occlusion
I. Ambient occlusion + bent normal




*A variety of lighting models implemented with GLSL shaders.* From the *OpenGL Shading Language, 2nd. Ed.* 2006