CS448h

Domain-Specific Languages for Graphics, Imaging, and Beyond

Pat Hanrahan
Jonathan Ragan-Kelley
Zach DeVito

Fall 2015

The Road to Point Reyes
Lucasfilm 1984

R.E.Y.E.S = Renders Everything You Ever Saw

Pixar Image Computer -> Reyes Machine

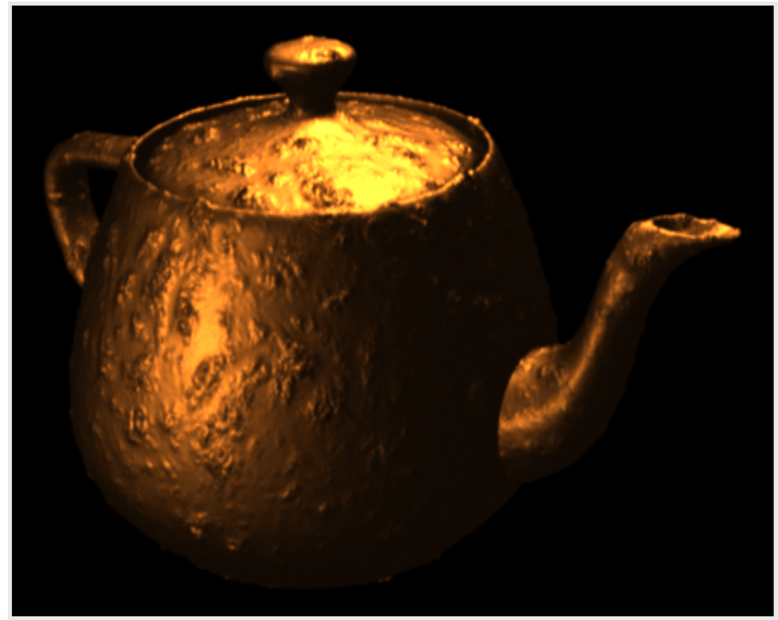
Challenge:

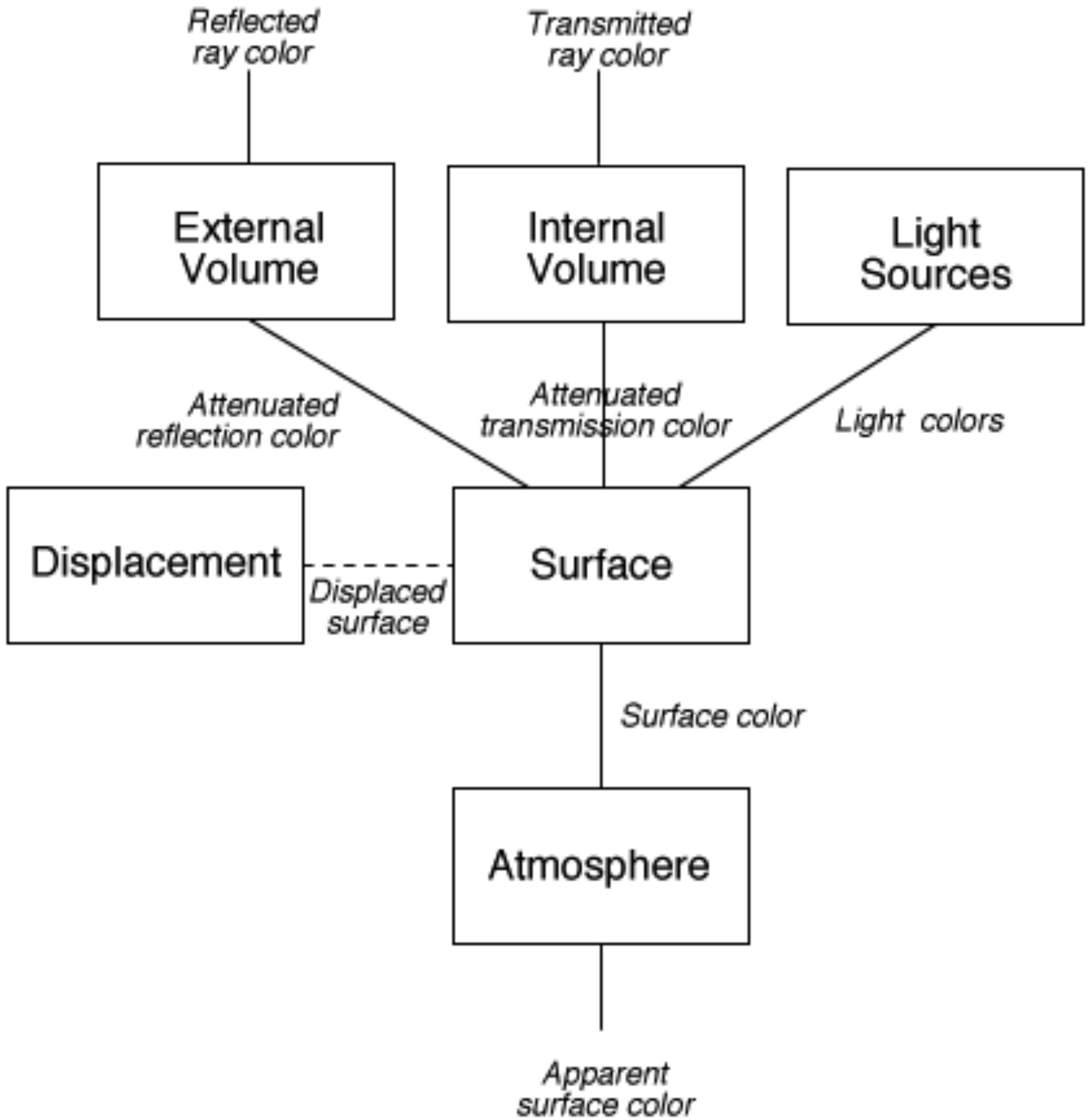
- Diverse geometric primitives
- Diverse materials and lighting

How to add new shaders?

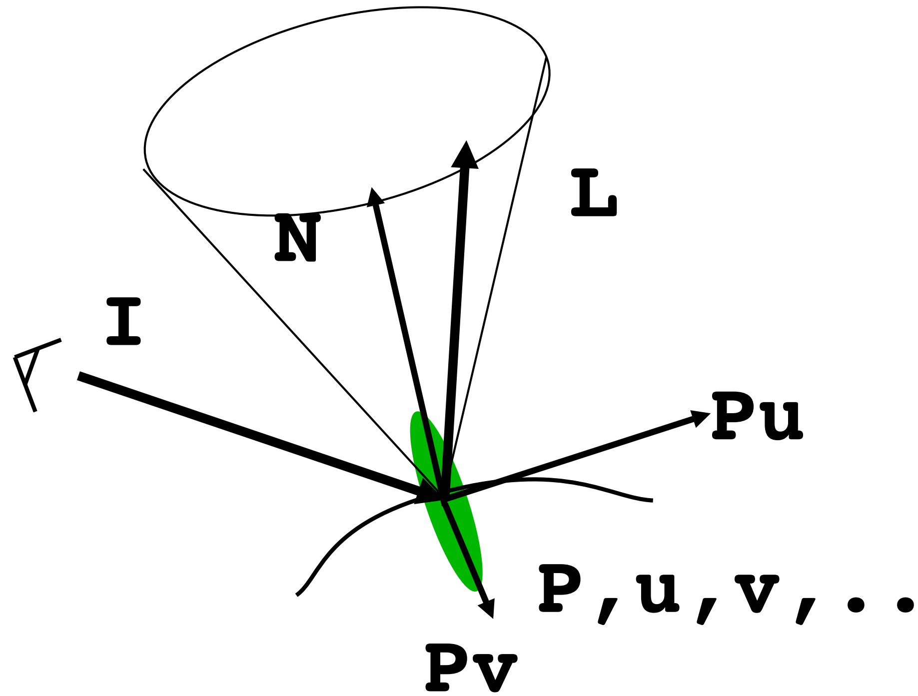
- Parameterized texture mapping pipe
- Shade trees expression language
- Procedural textures / Texture “mapping”

```
surface corrode(float Ks=0.4, Ka=0.1, rough=0.25) {
  float i, freq=1, turb=0;
  // compute fractal texture
  for( i=0; i<6; i++ ) {
    turb+=1/freq*noise(freq*P);
    freq*=2;
  }
  // perturb surface
  P -= turb * normalize(N);
  N = faceforward(normalize(calculatenormal(P)));
  // compute reflection and final color
  Ci = Cs*(Ka*ambient()+Ks*specular(N,I,rough));
}
```




Reflected ray color
Transmitted ray color
External Volume
Internal Volume
Light Sources
Attenuated reflection color
Attenuated transmission color
Light colors
Displacement
Surface
Displaced surface
Surface color
Atmosphere
Apparent surface color

Surface Geometry

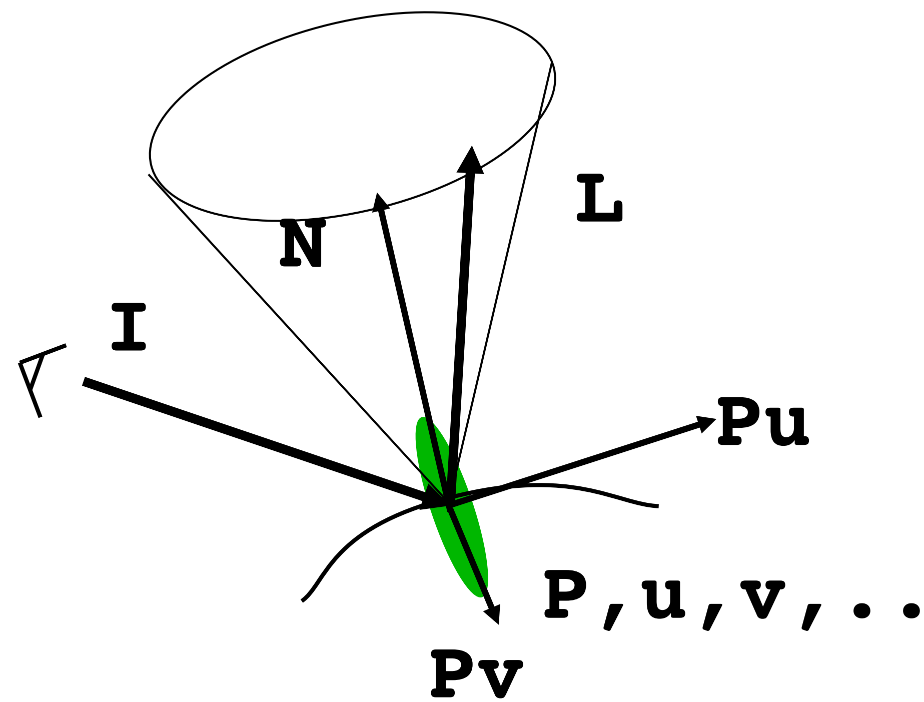


Material

```
Nn = normalize(N);
illuminance( P, Nn, PI/2 ) {
    Ln = normalize(L);
    Ci += Cs * Cl * Ln.Nn;
}
```

Surface Geometry

Light




```
illuminate( P, N, beamangle )
    Cl = (intensity*lightcolor)/ (L.L)

solar( D, 0 )
    Cl = intensity*lightcolor;
```

DSLs are Common!

Succinct Notation

`printf` format string

```
printf("%5.2f", floatvalue );

//width=5, precision=2
```

regular expressions (regex)

```
[-+]?([0-9]*\.[0-9]+|[0-9]+)
```

```
CFLAGS = -nostdlib -nostartfiles -ffreestanding

blink.bin: blink.c
	arm-none-eabi-gcc $(CFLAGS) blink.c -o blink.o
	arm-none-eabi-objcopy blink.o -O binary blink.bin
	arm-none-eabi-objdump -d blink.o > blink.list

install: blink.bin
	rpi-install.py blink.bin

clean :
	rm -f *.o *.bin *.list
```

```
%.d : %.c
	@set -e; rm -f $@; $(CC) -M $(CPPFLAGS) $< > $@.$$$$; \
	sed 's,\($*\)\.o[ :]*,\1.o $@ : ,g' < $@.$$$$ > $@; \
	rm -f $@.$$$$
```

```
export HOST=`hostname`
# if running bash
if [ -n "$BASH_VERSION" ]; then
    # include .bashrc if it exists
    if [ -f "$HOME/.bashrc" ]; then
        . "$HOME/.bashrc"
    fi
fi
# set PATH so it includes user's private bin if it exists
PATH=/bin:/usr/bin:/sbin:/usr/sbin
PATH=/Users/hanrahan/Courses/cs348b/pbrt-v2/src/bin:${PATH}
PATH=/usr/local/CrossPack-AVR/bin:${PATH}
PATH=/Applications/Arduino.app/Contents/Resources/Java/hardware/tools/avr/bin:${PATH}
PATH=/usr/local/bin:/usr/local/sbin:${PATH}
if [ -d "$HOME/bin" ] ; then
    PATH="$HOME/bin/clang/bin:$PATH"
    PATH="$HOME/bin:$PATH
fi
```

Other DSLs ...

awk, sed, ...

pic, tbl, eqn, ...

matlab

R

SQL

Why DSLs?

1. Productive

What are the 3 Biggest Ideas in Computer Science?

Abstraction
Abstraction
Abstraction

P. Hudak

DSL Abstractions

Easier to design for a well-defined domain

- start with an commonly used abstraction

Easier to use (less code, less time)

- abstraction matches programmer's mental model of the domain
- provide concise notation

Easier to specify and reason about

- use abstractions with formally specified semantics

```
class Vector { // pbrt Vector class
public:
    Vector() { x = y = z = 0.f; }
    Vector(float xx, float yy, float zz);
    Vector operator+(const Vector &v) const ;
    Vector& operator+=(const Vector &v) ;
    Vector operator-(const Vector &v) const ;
    Vector& operator-=(const Vector &v) ;
    Vector operator*(float f) const ;
    Vector &operator*=(float f) ;
    Vector operator/(float f) const ;
    Vector operator-() const { return Vector(-x, -y, -z); }
    float operator[](int i) const;
    float &operator[](int i);
    float LengthSquared() const { return x*x + y*y + z*z; }
    float Length() const { return sqrtf(LengthSquared()); }
    explicit Vector(const Normal &n);
    bool operator==(const Vector &v) const ;
    bool operator!=(const Vector &v) const ;
}
```

```
Spectrum Microfacet::f(
    const Vector &wo,
    const Vector &wi) const
{
    float cosThetaO = AbsCosTheta(wo);
    float cosThetaI = AbsCosTheta(wi);
    if (cosThetaI == 0.f || cosThetaO == 0.f)
        return Spectrum(0.f);
    Vector wh = wi + wo;
    if (wh.x == 0. && wh.y == 0. && wh.z == 0.)
        return Spectrum(0.f);
    wh = Normalize(wh);
    float cosThetaH = Dot(wi, wh);
    Spectrum F = fresnel->Evaluate(cosThetaH);
    return R * distribution->D(wh)
             * G(wo, wi, wh) * F
             / (4.f * cosThetaI * cosThetaO);
}
```

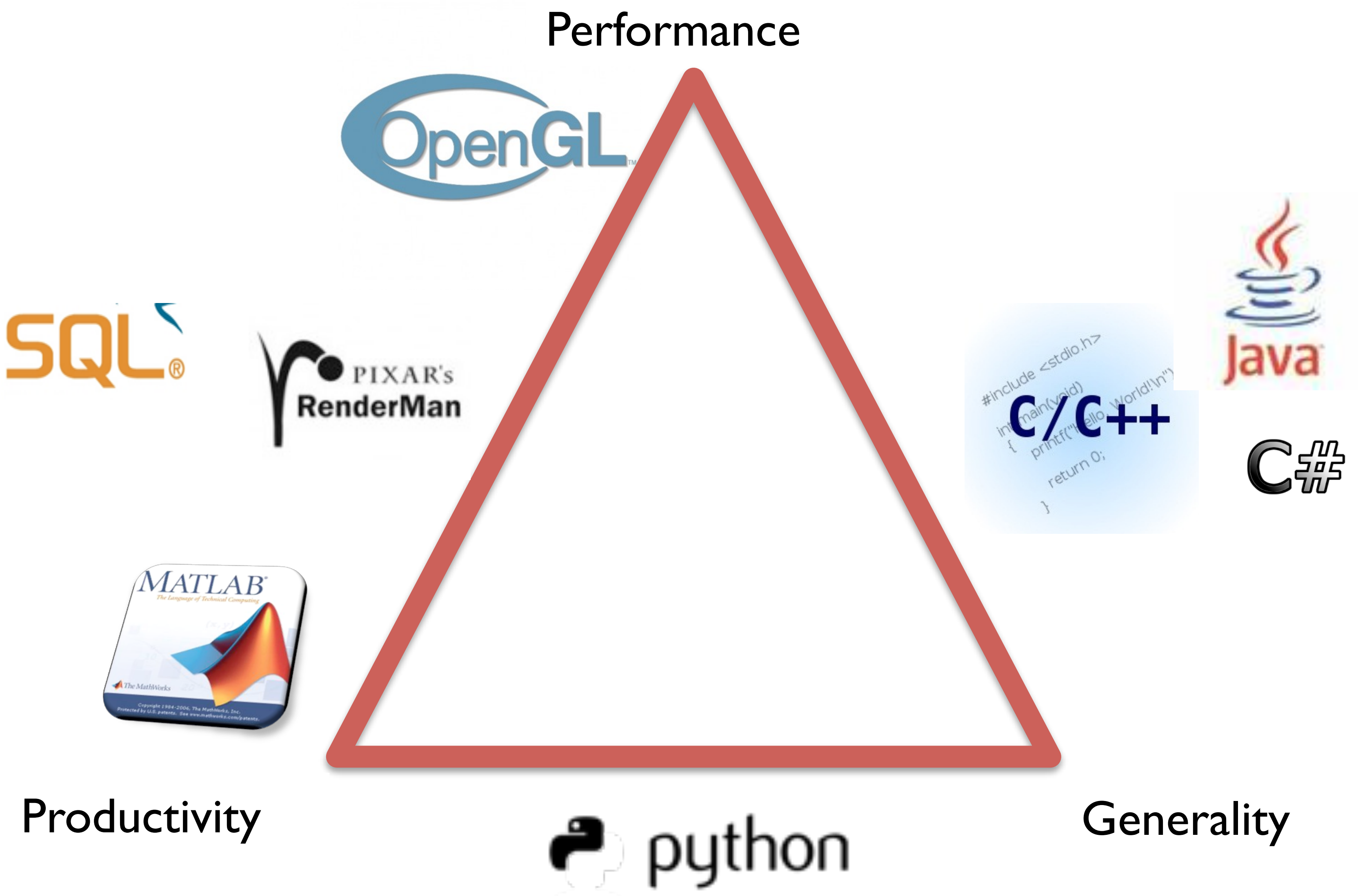
Little Languages

Jon Bentley, CACM 29(8), 1986

Defining "little" is harder; it might imply that the first-time user can use this system in an hour or master the language in a day, or perhaps the first implementation took just a few days. In any case, a little language is specialized to a particular problem domain and does not include many features found in conventional languages.

Why DSLs?

1. Productive
2. Performant


Performance
OpenGL
SQL
PIXAR's
RenderMan
MATLAB
C/C++
Java
C#
Productivity
python
Generality

Optimizations

"General-purpose" optimizers do not always work in practice since optimization is fundamentally a hard search problem

The abstractions in the domain provide transformations of the code that enable high-level optimization

Matrix multiplication is associative

- `(A*B)*C = A*(B*C)`

Multiply `[lxm]*[mxn]=l*m*n` ops

- `[4x4]*[4x4]=64` ops
- `[4x4]*[4x1]=16` ops

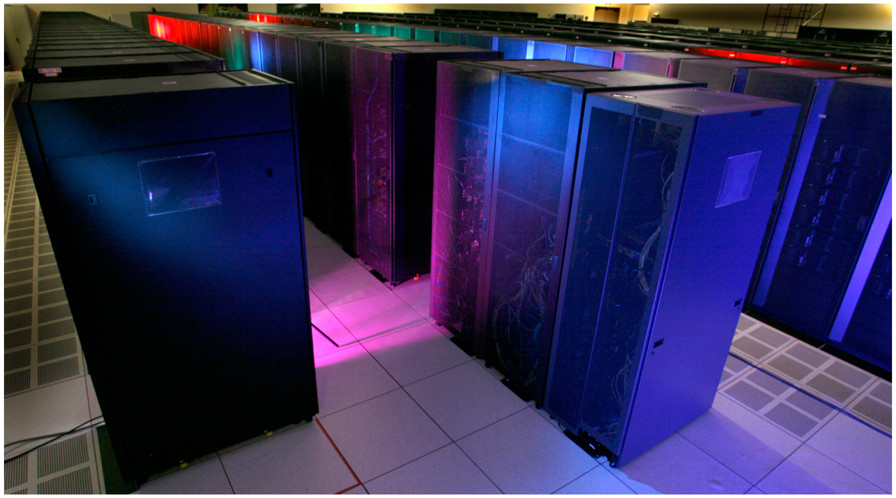
Search to find the best multiplication order

- `T1*T2*p=` ?

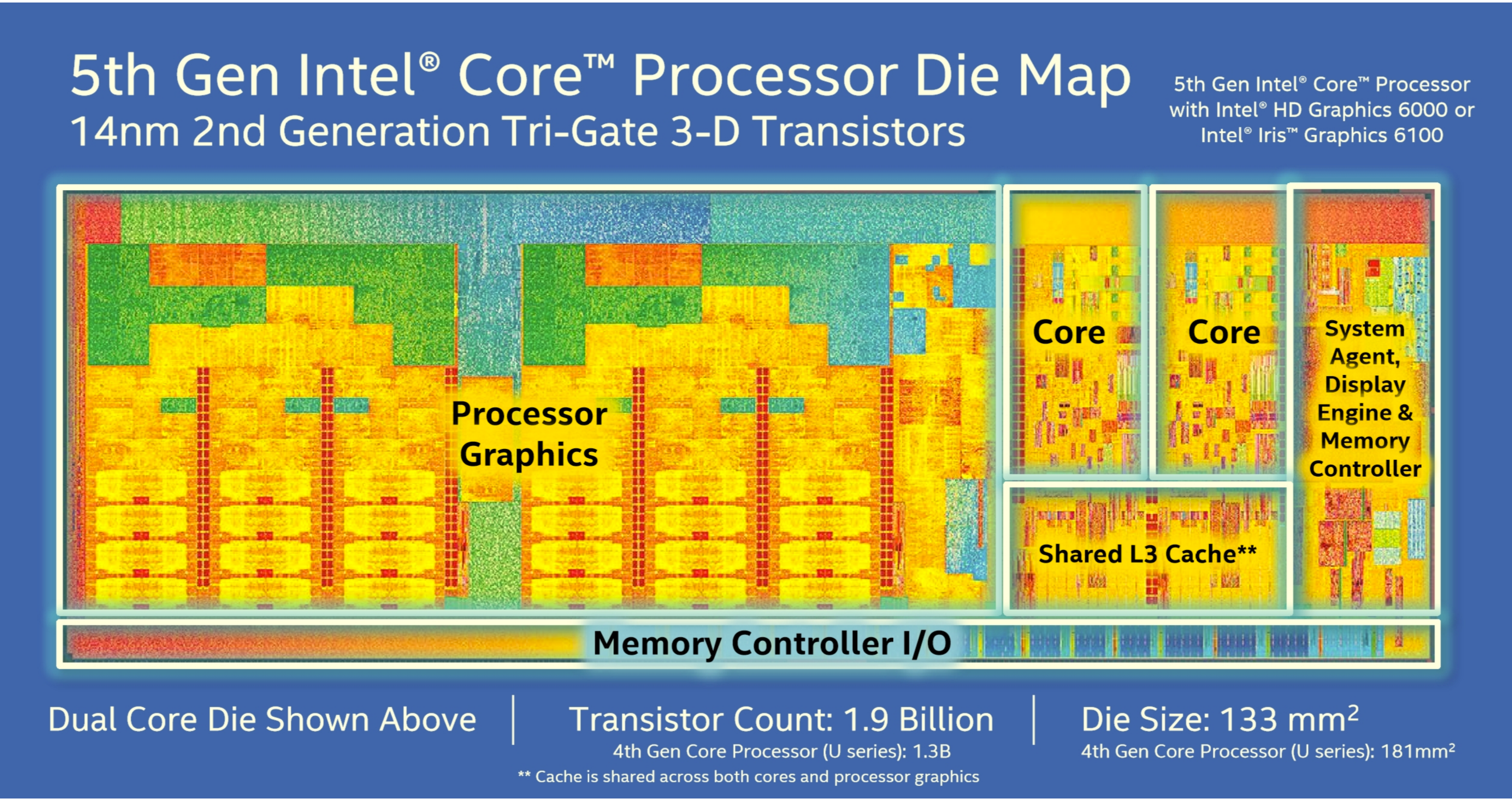
Power Efficiency

Mobile

Cloud

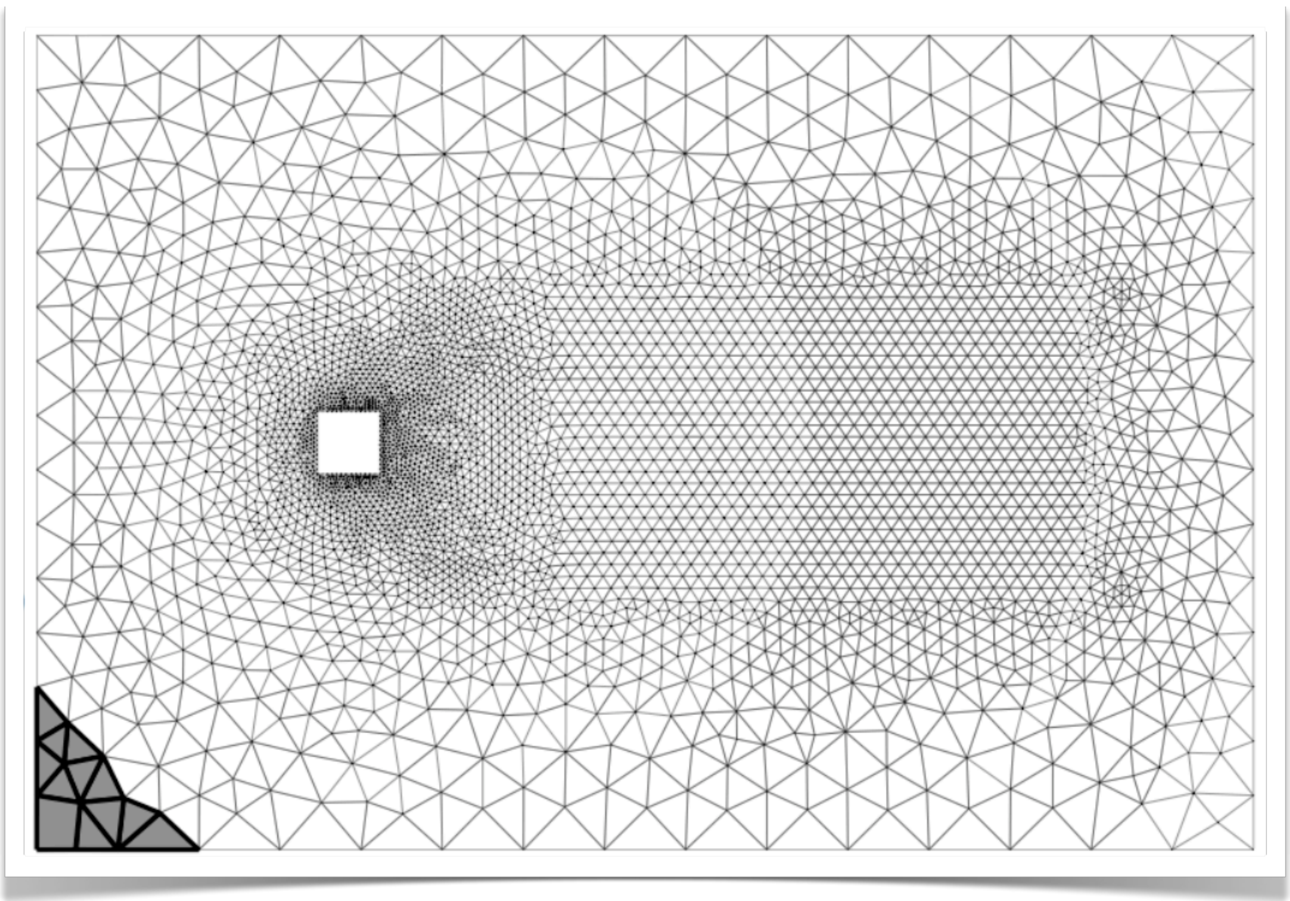


CPU + GPU


5th Gen Intel® Core™ Processor Die Map
14nm 2nd Generation Tri-Gate 3-D Transistors
5th Gen Intel® Core™ Processor with Intel® HD Graphics 6000 or Intel® Iris™ Graphics 6100
Processor Graphics
Core
Core
System Agent, Display Engine & Memory Controller
Shared L3 Cache**
Memory Controller I/O
Dual Core Die Shown Above
Transistor Count: 1.9 Billion
4th Gen Core Processor (U series): 1.3B
Die Size: 133 mm²
4th Gen Core Processor (U series): 181mm²
** Cache is shared across both cores and processor graphics

DSLs enable Parallelism

Well-known strategies for parallelizing programs in different domains



DSL Enable Specialized Hardware

Expensive computational units can be built-in

- e.g. texture mapping hardware

Calculations that can't be optimized automatically can be disallowed

- e.g. rasterization / ray tracing is hard

How Many DSLs are There?

How Many DSLs are There?

How Many Libraries?

Libraries vs Languages

Languages provide rules of composition : form sentences from nouns and verbs

Languages can be transformed / compiled into other languages : the compiler version can be executed, can be faster

Libraries and frameworks emphasis interoperability

- "The goal of a language is to make it possible to build the most powerful libraries"
- Said another way, provide mechanisms for building the best possible abstractions
- Can we have the best of both worlds?

Research DSLs

Liszt (Gilbert, ...) and Simit (J, ...)

- Simulation with meshes and particles

Darkroom (James, Z, J) / Halide (J)

- Image processing

Church / Quicksand (Daniel, ...)

- Procedural modeling using probabilistic programming language

Terra

A low-level counterpart to Lua

Download **TAR Ball** | View On **GitHub**

Getting Started

API Reference

Publications

Zach DeVito
zdevito at stanford dot edu

Terra is a new low-level system programming language that is designed to interoperate seamlessly with the **Lua** programming language:

```
-- This top-level code is plain Lua code.
print("Hello, Lua!")

-- Terra is backwards compatible with C
-- we'll use C's io library in our example.
C = terralib.includec("stdio.h")

-- The keyword 'terra' introduces
-- a new Terra function.
terra hello(argc : int, argv : &rawstring)
    -- Here we call a C function from Terra
    C.printf("Hello, Terra!\n")
    return 0
end

-- You can call Terra functions directly from Lua
hello(0,nil)

-- Or, you can save them to disk as executables or .o
-- files and link them into existing programs
terralib.saveobj("helloterra",{ main = hello })
```

Like C, Terra is a simple, statically-typed, compiled language with manual memory management. But unlike C, it is designed from the beginning to interoperate with Lua. Terra functions are first-class Lua values created using the `terra` keyword. When needed they are JIT-compiled to machine code.

You can **use** Terra and Lua as…

A scripting-language with high-performance extensions. While the

Logistics

- 8 lectures
- 4 case studies
- 2 programming assignments
- project
  - proposal
  - final presentation
  - final write-up in the form of a paper
- critical thinking: how to read a research paper
- critical thinking: debate