CSE 167 (FA22) Computer Graphics: Closing Remark

Albert Chern
Other graphics courses in UCSD

- CSE 168: Computer Graphics II Rendering
- CSE 169: Computer Animation
- CSE 270 (to appear): Discrete Differential Geometry
- CSE 272: Advanced Image Synthesis
- CSE 273: Computational Photography
- CSE 274: Topics in Computer Graphics
- CSE 290: Seminars in Graphics
- CSE 291: Physical Simulations
Broader impacts of computer graphics

- Programming languages
- Mechanical engineering
- Nuclear engineering
- Robotics
- Virtual reality
- Scientific visualization
- Human computer interaction
- Computer architecture
- Medical imaging
- Differential geometry
- Manufacturing
- Art
- Cognitive science
- Computer-aided design
- Computational imaging
- Cultural heritage preservation
- Machine learning
- Bayesian statistics
- Medical imaging
- Computer architecture
- Human computer interaction
- Scientific visualization
- Virtual reality
- Programming languages
- Mechanical engineering
- Nuclear engineering
- Robotics
- Differential geometry
- Manufacturing
- Art
- Cognitive science
- Computer-aided design
- Computational imaging
- Cultural heritage preservation
- Machine learning
- Bayesian statistics
Modeling and processing visual data
Processing perceptual data

visual  audio  tactile  scent
Simulating reality with code

99 lines C++ code, no dependencies from Kevin Beason
Simulating reality with code

```
from functools import reduce
from math import factorial
import numpy as np
import scipy.sparse
from scipy.ndimage import map_coordinates
from scipy.sparse.linalg import factorized

def differences(accuracy, order):
    def parts(points, o):
        coefficients = np.vander(points)
        coefficients = np.linalg.inv(coefficients)
        return coefficients[-o - 1] * factorial(o)
    return tuple(parts(range(-accuracy, accuracy + 1), o) for o in order)

def matrices(shape, operators, combine):
    def parts():
        for i, o in enumerate(operators):
            diagonals = []
            for j, p in enumerate(o):
                index = j - len(o) // 2
                diagonals.append((p * np.ones(shape[i] - abs(index)), index))
            matrix = scipy.sparse.diags(*zip(*diagonals))
            if combine:
                yield matrix
            else:
                yield reduce(sp.kron, (matrix if k == i else sp.identity(d) for k, d in enumerate(shape)));
    return reduce(lambda a, b: sp.kronsum(b, a), parts()) if combine else tuple(parts())

class Fluid:
    def __init__(self, shape, viscosity, quantities):
        self.shape = shape
        self.size = np.product(shape)
        self.dimensions = len(shape)
        self.viscosity = viscosity
        self.quantities = {}
        for q in quantities:
            self.quantities[q] = np.zeros(self.size)
        self.velocity_field = np.zeros((self.size, self.dimensions))
        self.indices = np.dstack(np.indices(self.shape)).reshape(self.size, self.dimensions)
        self.gradient = matrices(shape, differences(1, (1,) * self.dimensions), False)
        laplacian = matrices(shape, differences(1, (2,) * self.dimensions), True)
        self.pressure_solver = factorized(laplacian)
        self.viscosity_solver = factorized(sp.identity(self.size) - laplacian * viscosity)

    def advect_diffuse(self):
        advection_map = np.moveaxis(self.indices - self.velocity_field, -1, 0)
        for d in range(self.dimensions):
            self.velocity_field[..., d] = kernel(self.velocity_field[..., d])
        for k, q in self.quantities.items():
            self.quantities[k] = kernel(q)

    def project(self):
        divergence = sum(self.gradient[d].dot(self.velocity_field[..., d]) for d in range(self.dimensions))
        pressure = self.pressure_solver(divergence)
        for d in range(self.dimensions):
            self.velocity_field[..., d] -= self.gradient[d].dot(pressure)
```

54 lines Python w/ scipy
from Gregory Johnson
Main subfields of graphics

- geometry
- animation
- rendering

Subfields:
- Scattering
  - Caustic
  - Specular reflection
  - Indirect light
  - Volumentric scattering
- Microstructure
- Direct light
- Shadow
- Defocus blur
More subfields

- sound simulation
- computational photography/imaging
- 3D design, fabrication
- user interface
- tactile rendering
- visualization
Graphics for fun

how to peel an orange

how to assemble a 3D model using LEGO blocks

how to cut a bunny into a Rubik’s cube
Broader impacts of computer graphics

- virtual reality
- scientific visualization
- human computer interaction
- computer architecture
- medical imaging
- differential geometry
- manufacturing
- art
- cognitive science
- Bayesian statistics
- machine learning
- computational imaging
- cultural heritage preservation
- computer-aided design
- robotics
- nuclear engineering
- mechanical engineering
- programming languages
Broader impacts of computer graphics

- Graphics is applied everywhere
- Graphics drives the development of programming language / UI / hardware
- Graphics lead to new insights
Broader impacts of computer graphics

- Graphics is applied everywhere
- Graphics drives the development of programming language / UI / hardware
- Graphics lead to new insights
Movies and Video Games

“The Adventures of Tintin”, 2011

“Assassin’s Creed Odyssey”, 2018
Physically accurate optics

- motion blur
- fur materials
- shadow
- global illumination
- caustics
- volumetric scattering
- specular reflection
at 2014, around 60-75% of all IKEA’s product images (images showing only a single product) are CG. 35% of non-produce images are fully CG.

these are all computer generated!!
Graphics in architectural design

- Daylighting simulation
- uses a classical rendering algorithm “irradiance caching” [Ward 1992]

from Nathaniel Jones
• applied procedural modeling to synthesize structurally sound buildings

“Procedural Modeling of Structurally-Sound Masonry Buildings”, Emily Whiting et al.
Graphics in architectural design
Graphics for manufacturing

“Computational Multicopter Design”, Tao Du et al.


“CurveBoards: Integrating Breadboards into Physical Objects to Prototype Function in the Context of Form”, Junyi Zhu et al.
• high-dynamic range imaging
Computational photography

- content-aware resizing
Computational photography

- photomontage

“Interactive Digital Photomontage”, Agarwala et al.
Cultural heritage preservation

- digital Michelangelo project (1999)

rendering made by Henrik Jensen
Cultural heritage preservation

• recovering ancient artifacts

Figure 1: Reassembling a gargoyle statue: photo (bottom left) and 3D models (top left) of the fragments, final assembly (right).

Figure 12: Reassembling a fractured Venus model.

“Reassembling Fractured Objects by Geometric Matching”, Qi-Xing Huang et al.
Scientific visualization

- visualizing vorticity information in fluids

“Inside Fluids: Clebsch Maps for Visualization and Processing”, Albert Chern et al.
Visualizing medical data

- extracting isosurfaces from CT/MRI scans

Volume rendering
Volume rendering

computer graphics ~≈≈ nuclear engineering ~≈≈ biomedical optics
Cognitive science

- computer graphics for synthesizing visual stimuli

“Illumination-induced Apparent Shift in Orientation of Human Heads”, Nikolaus Troje et al.
• vision algorithms based on inverse graphics fit better to biological data

IT Cortex: neural responses from macaque’s brain
EIG: inverse graphics network
VGG: classification network

"Efficient inverse graphics in biological face processing", Ilker Yildirim et al.
• graphics models are often used to build human perceptual models

these are psychology papers!!

“Perception of Surface Curvature and Direction of Illumination From Patterns of Shading”
“The Perception of Cast Shadows”
“Perceptual Biases in the Interpretation of 3D Shape from Shading”
“The Visual Perception of Metal”
Graphics reconstructs/simulates real world

- this leads to myriad of real-world applications
Broader impacts of computer graphics

- Graphics is applied everywhere
- Graphics drives the development of programming language / UI / hardware
- Graphics lead to new insights
Broader impacts of computer graphics

- Graphics is applied everywhere
- Graphics drives the development of programming language / UI / hardware
- Graphics lead to new insights
From punched card to sketchpad
• How do we talk to computers through drawing?

https://www.youtube.com/watch?v=495nCzxM9PI
• How do we talk to computers through drawing?

Figure 16. Winking girl, “Nefertite,” and her component parts.
Sketchpad was the first object-oriented programming language.

Figure 5. Generic structure. The n-component elements for each point or line, etc., are collected under the generic blocks “lines,” “points,” etc., shown.
Sketchpad inspires Smalltalk

Ivan Sutherland (Utah Professor)

sketchpad

inherits

Alan Kay (Utah PhD)

smalltalk

inherits
As graphics move to 3D...

- how do we describe the shape of objects?
- how do we describe the reflective property?
- how do we describe patterns?
Programming language for shading

• Procedural and symbolic description of material

Figure 4. Road to Point Reyes.

Figure 1a. Shade tree for copper.

“Shade Trees”, Cook 1984
Programming language for shading

• Procedural and symbolic description of material

```
function boring_marble(point)
    x = point[1]
    return marble_color(sin(x))
```

```
function marble(point)
    x = point[1] + turbulence(point)
    return marble_color(sin(x))
```

```
function turbulence(p)
    t = 0
    scale = 1
    while (scale > pixelsize)
        t += abs(Noise(p / scale) * scale)
        scale /= 2
    return t
```

“An Image Synthesizer”, Perlin 1985
• Procedural and symbolic description of material
Programming language for shading

- Pixar's RenderMan Shading Language

```
program


diffuse = KA * light;
diffuse = diffuse * Kd * N.

specular = specular_difffuse_light(light, d, N);

diffuse += specular;
```

Figure 1. Surface shader state.

Figure 4. Corroded teapot.

"A Language for Shading and Lighting Calculations", Hanrahan and Lawson, 1990
Shading = evaluating grids

- easily parallelizable and we can build hardware of it
- graphics popularizes SIMD architectures!

https://nyu-cds.github.io/python-gpu/01-introduction/
Shading -> general-purpose GPU

```

kernel void saxpy (float a, float4 x<> , float4 y<> ,
  out float4 result<> ) {
  result = a*x + y;
}

void main (void) {
  float a;
  float4 X[100], Y[100], Result[100];
  float4 x<100>, y<100>, result<100>;
  ... initialize a, X, Y ...
  streamRead(x, X);    // copy data from mem to stream
  streamRead(y, Y);
  saxpy(a, x, y, result);  // execute kernel on all elements
  streamWrite(result, Result); // copy data from stream to mem
}

```
sparse matrix multiply, molecular dynamics, ... etc

“Brook for GPUs: Stream Computing on Graphics Hardware”, Ian Buck et al. (SIGGRAPH 2004)
kernel void saxpy (float a,
    out float result = a*x + y;
}

void main (void) {
    float a;
    float4 X[100], Y[100], Re
    float4 x<100>, y<100>, re
    ... initialize a, X, Y ...
    streamRead(x, X);
    streamRead(y, Y);
    saxpy(a, x, y, result);
    streamWrite(result, Result

“Brook for GPUs: Stream Computing on Graphics Hardware”, Ian Buck et al. (SIGGRAPH 2004)
Broader impacts of computer graphics

• Graphics is applied everywhere

• Graphics drives the development of programming language / UI / hardware

• Graphics lead to new insights
Broader impacts of computer graphics

- Graphics is applied everywhere
- Graphics drives the development of programming language / UI / hardware
- Graphics lead to new insights
Vision as inverse graphics

1. #include <math.h>
   
2. #include <stdlib.h>
3. #include <stdio.h>
4. 
5. struct Vec {
6.     double x, y, z;
7.     Vec(double x_=0, double y_=0, double z_=0){ x=x_; y=y_; z=z_; }
8.     Vec operator+(const Vec &b) const { return Vec(x+b.x,y+b.y,z+b.z); }
9.     Vec operator-(const Vec &b) const { return Vec(x-b.x,y-b.y,z-b.z); }
10.    Vec operator*(double b) const { return Vec(x*b,y*b,z*b); }
11.    Vec mult(const Vec &b) const { return Vec(x*b.x,y*b.y,z*b.z); }
12.    Vec& norm(){ return *this *= (1/sqrt(x*x+y*y+z*z)); }
13.    double dot(const Vec &b) const { return x*b.x+y*b.y+z*b.z; }
14. }
15. 
16. struct Ray { Vec o, d; Ray(Vec o_, Vec d_) : o(o_), d(d_) {} }
17. 
18. enum Refl_t { DIFF, SPEC, REFR }; // material types, used in radiance()
19. 
20. struct Sphere {
21.     double rad; // radius
22.     Vec p, e, c; // position, emission, color
23.     Refl_t refl; // reflection type (DIFFuse, SPECular, REFRactive)
24.     Sphere(double rad_, Vec p_, Vec e_, Vec c_, Refl_t refl_): rad(rad_), p(p_), e(e_), c(c_), refl(refl_) {}
25.     double intersect(const Ray &r) const {
26.         Vec op = p-r.o;
27.         if(det<0) return 0;
28.         det=sqrt(det);
29.         return (t=b-det)>eps ? t : ((t=b+det)>eps ? t : 0);
30.     }
31. }
32. 
33. Sphere spheres[] = {
34.     Sphere(1e5, Vec( 1e5+1,40.8,81.6), Vec(), Vec(.75,.25,.25),DIFF), //Left
35.     Sphere(1e5, Vec(-1e5+99,40.8,81.6), Vec(), Vec(.25,.25,.75),DIFF), //Rght
36.     Sphere(1e5, Vec(50,40.8, 1e5), Vec(), Vec(.75,.75,.75),DIFF), //Back
37.     Sphere(1e5, Vec(50,40.8,-1e5+170), Vec(), Vec(), DIFF), //Frnt
38.     Sphere(1e5, Vec(50, 1e5, 81.6), Vec(), Vec(.75,.75,.75),DIFF), //Botm
39.     Sphere(1e5, Vec(50,-1e5+81.6,81.6), Vec(), Vec(.75,.75,.75),DIFF), //Top
40.     Sphere(16.5, Vec(27,16.5,47), Vec(), Vec(1,1,1)*.999, SPEC), //Mirr
41.     Sphere(16.5, Vec(73,16.5,78), Vec(), Vec(1,1,1)*.999, REFR), //Glas
42.     Sphere(600, Vec(50,681.6-.27,81.6), Vec(12,12,12), Vec(), DIFF) //Lite
43. }
44. 
45. inline double clamp(double x){ return x<0 ? 0 : x>1 ? 1 : x; }
46. inline int toInt(double x){ return (int)pow(clamp(x),1/2.2)*255+.5; }
47. 
48. inline bool intersect(const Ray &r, double &t, int &id){
49.     double n=sizeof(spheres)/sizeof(Sphere), d, inf=t=1e20;
50.     for(int i=int(n);i--;)
51.         if((d=spheres[i].intersect(r))&&d<t){t=d;id=i;}
52.     return t<inf;
53. }
54. 
55. Vec radiance(const Ray &r, int depth, unsigned short *Xi){
56.     double t;
57.     int id=0;
58.     // gradient descent
Graphics and basic research

- A field usually has an obvious direction for follow-up research
- Basic research: freedom to think about fundamental questions
- In contrast to math/physics/engineering, graphics provides the freedom to rewrite physics
  - Motivation in graphics: Efficiency, robustness, simplicity, look beautiful
  - Often stumble upon new understanding or new science
Biological behavior dynamics

Craig W. Reynolds, SIGGRAPH 1987
Flocks, Herds, and Schools
Quantum mechanics

Schrödinger equation

Bohmian mechanics (fluid interpretation of QM)

kind of disproven

Schrödinger’s Smoke (a fluid animation solver for graphics)

Chern et al, SIGGRAPH 2016
Schrödinger’s Smoke
Quantum mechanics
"Ensemble-cio: Full-body dynamic motion planning that transfers to physical humanoids", Igor Mordatch et al. video from a talk by Emanuel Todorov (https://www.youtube.com/watch?v=7MwY8cK-1sQ)
Geometry -> better elastic rod model

Bergou, Wardetzky, Robinson, Audoly, Grinspun 2006
Discrete Elastic Rods
Disney Frozen -> snow & sand model

Digging character

“A material point method for snow simulation”, Alexey Stomakhin et al.
Geometry processing $\rightarrow$ discrete physics

modeling & analysis

- Invariants, Symmetries

discretization

Discrete equations

Exact rather than approximation

discrete differential geometry
Discrete wave absorbing layer

Chern 2019

Reflectionless discrete perfectly matched layer
Broader impacts of computer graphics

• Graphics is applied everywhere

• Graphics drives the development of programming language / UI / hardware

• Graphics lead to new insights
Broader impacts of computer graphics

- Programming languages
- Mechanical engineering
- Nuclear engineering
- Robotics
- Virtual reality
- Scientific visualization
- Human computer interaction
- Computer architecture
- Medical imaging
- Differential geometry
- Manufacturing
- Art
- Cognitive science
- Bayesian statistics
- Computer-aided design
- Computational imaging
- Cultural heritage preservation
- Machine learning