

CSCI 4560/6560 Computational Geometry

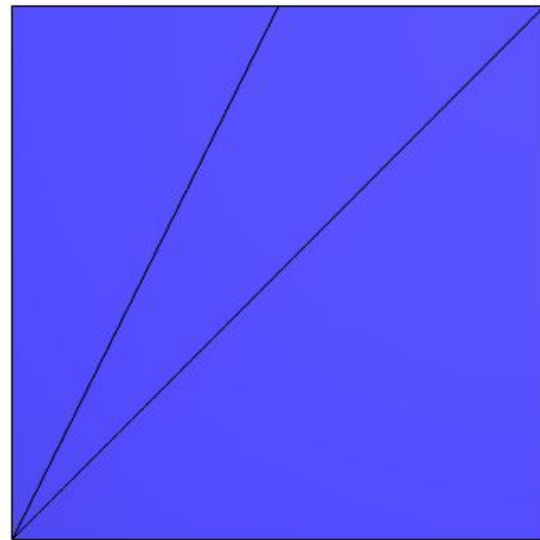
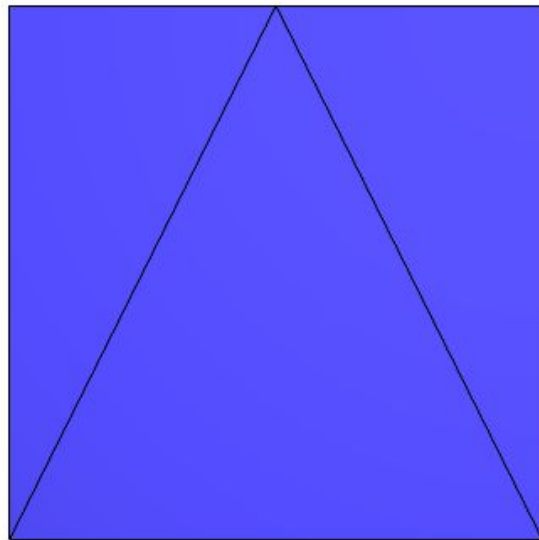
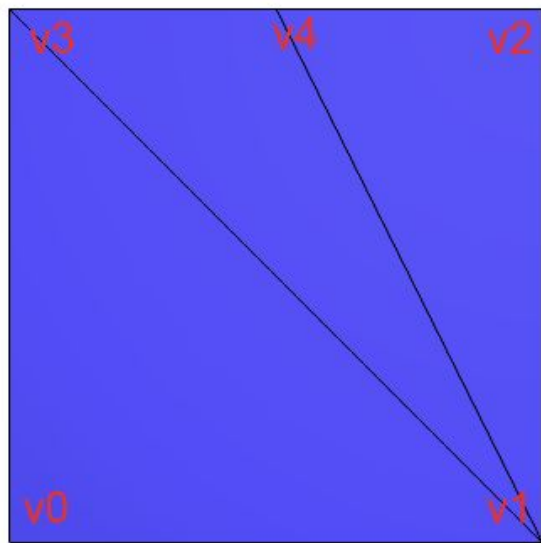
<https://www.cs.rpi.edu/~cutler/classes/computationalgeometry/F23/>

Lecture 19: General Position, Robustness & Exact Computation

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

Homework 7: Delaunay Triangulation Edge Flips



$v_1-v_3 \rightarrow v_0-v_4$
 $v_1-v_4 \rightarrow v_0-v_2$

Proposals due Monday Nov 6th

[Upload to Submitty](#)

Proposal

As you choose your topic and begin to flesh out the details, keep in mind that implementing new data structures or algorithms can take much longer than anticipated. Also be warned that designing and implementing even relatively simple user interfaces require a lot of effort (and is not particularly relevant to this course).

Your proposal should be formatted using pdf. The document should be a minimum of 500 words for an individual project (equivalent of 2 pages double spaced text) or 800 words for a team of two and include:

- A brief summary of the technical problem you are going to investigate.
- A list of the specific research papers and other sources you've collected for background reading. Talk with the instructor if you are unable to find at least 3 relevant academic references. Read and summarize the contributions of each reference and describe how your project relates to this work.
- A timeline for your assignment with a list of the tasks you will execute and *who will do what*. It's ok to list optional tasks that you will work on once the core features are complete. You will be graded relative to the completion of the core tasks, so make sure your plan is feasible.

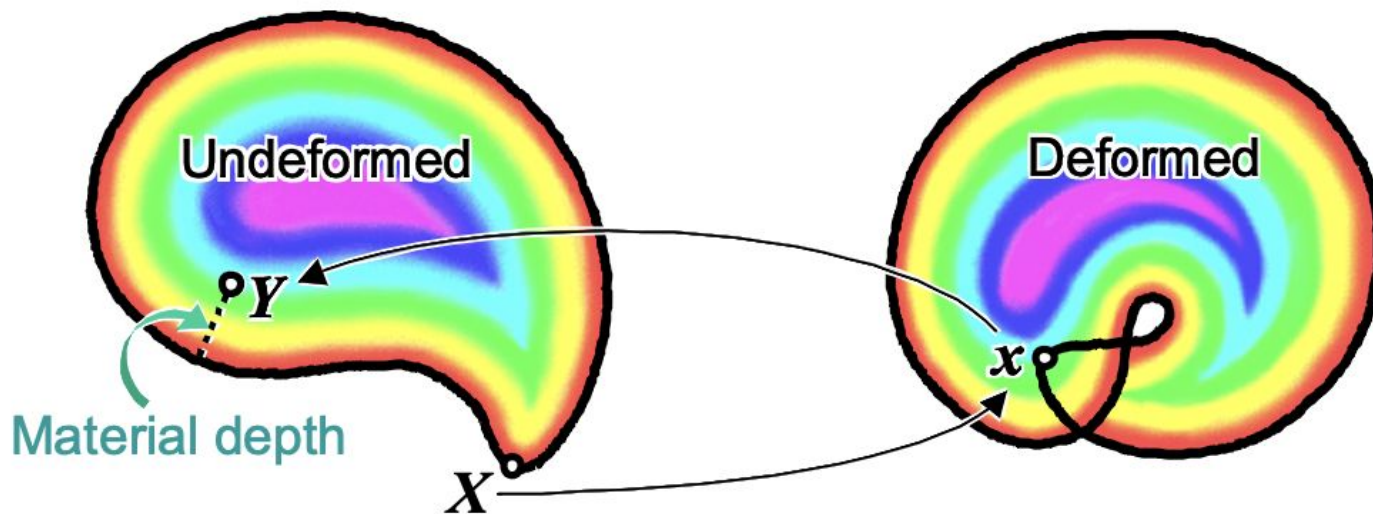
Outline for Today

- Homework 7 & Final Project Proposal Questions
- **Last Time: Signed Distance & Level Sets**
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

Motivation: Collision Detection

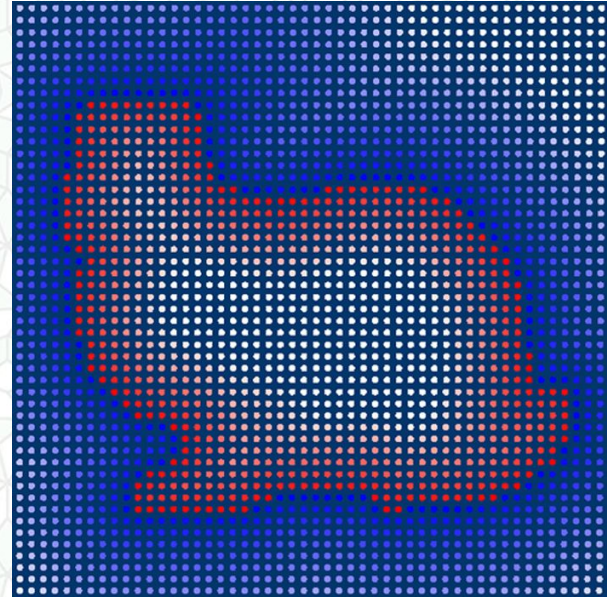
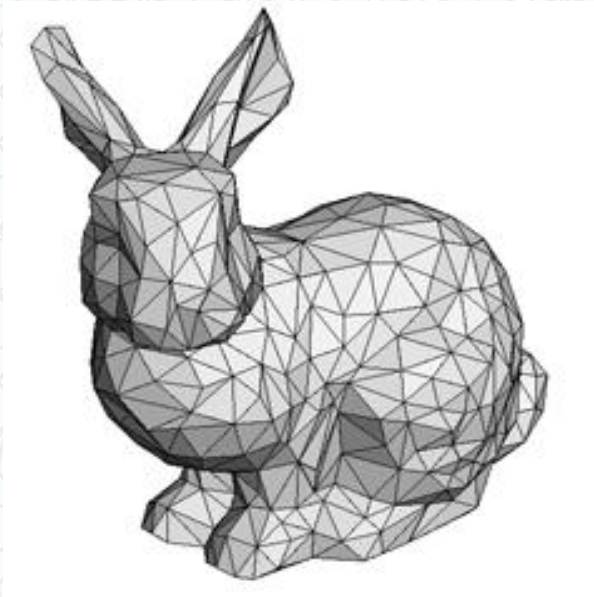
- Detect the intersection
- Depth of intersection penetration
- Gradient & normal of closest surface –
Determine penalty force to resolve collision

“An Implicit Finite Element Method
for Elastic Solids in Contact”,
Hirota, Fisher, State, Lee, & Fuchs,
SCA 2001



Explicit vs. Implicit Surface Representations

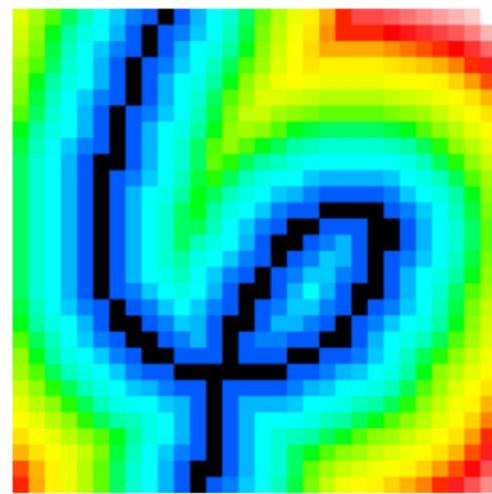
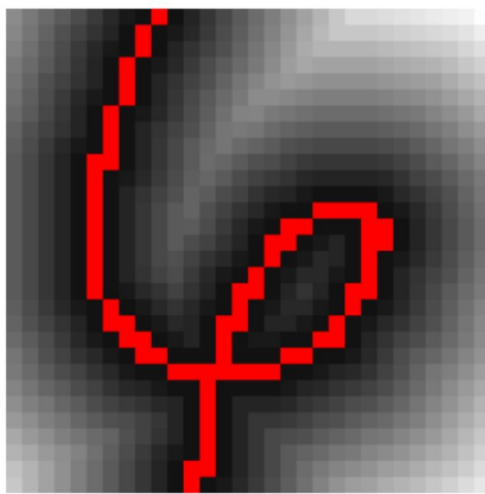
- We may not be able to construct a compact mathematical function...
- But can we convert the bunny mesh into a signed distance field?



Computing a Signed Distance Field

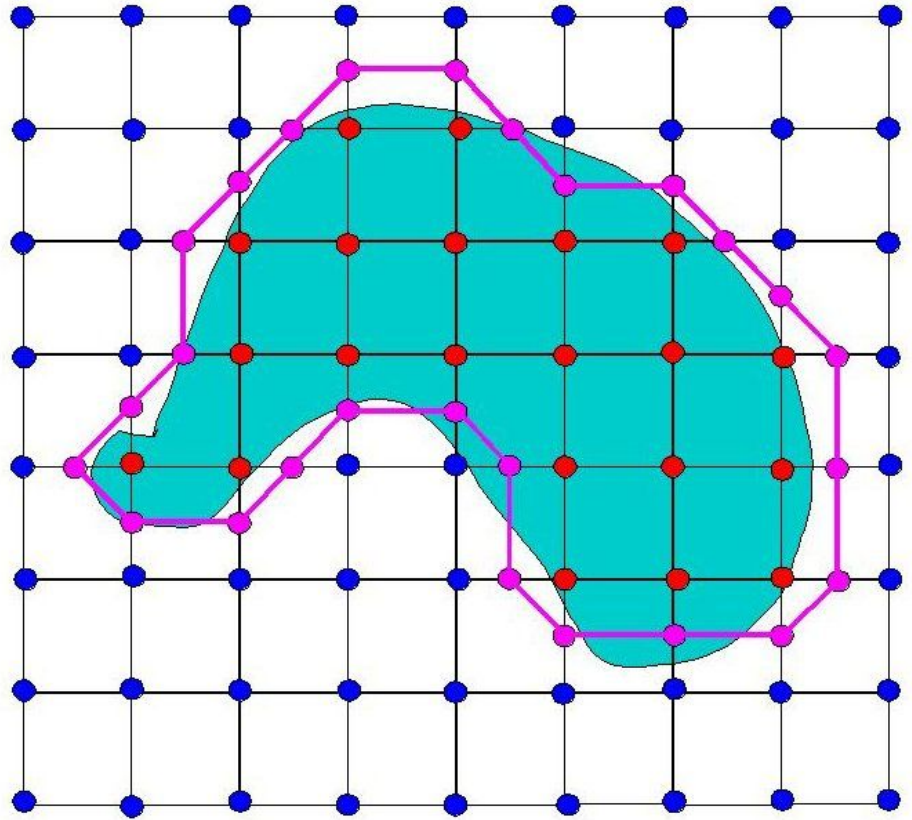
- Given a shape/surface
- Cost to compute shortest distance to original shape for each point (on a grid) in the volume?

*Naive: $O(\text{\# of volume grid samples} * \text{\# of surface elements}) = O(w^2h^2)$*



Marching Cubes

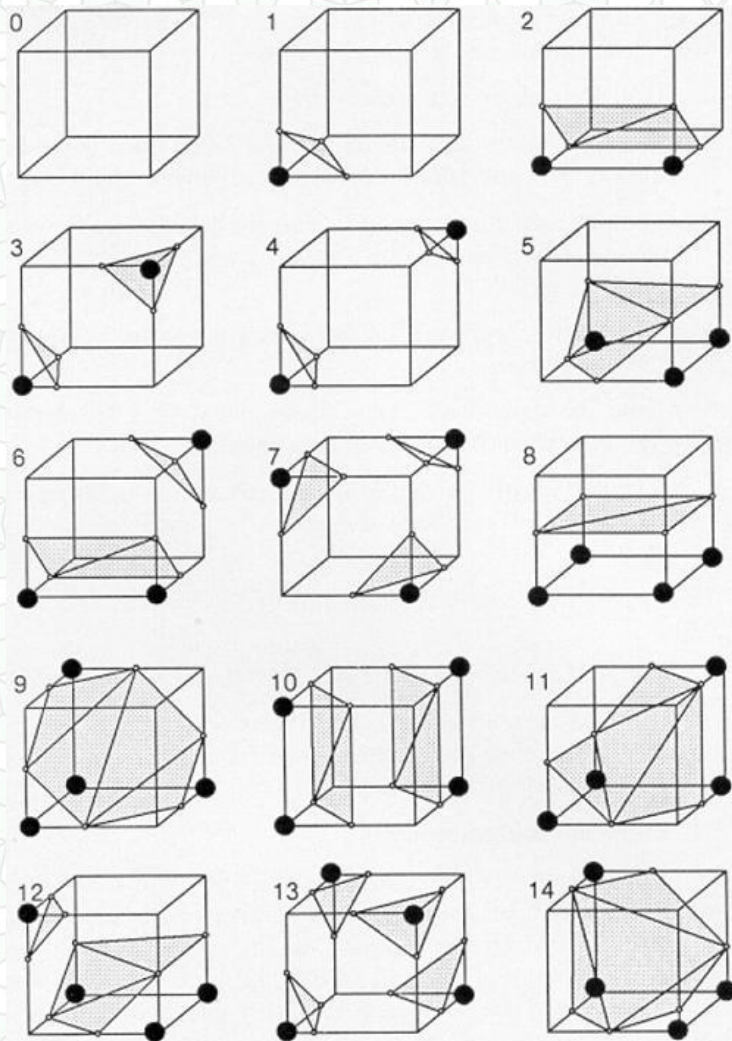
- Each point in the 3D grid is labeled “inside” (red dots) or “outside” (blue dots) the unknown surface.
- Any cell in the grid that has at least one red vertex and at least one blue vertex, must be crossed by the unknown surface.
- We can piecewise construct an approximation of the surface.

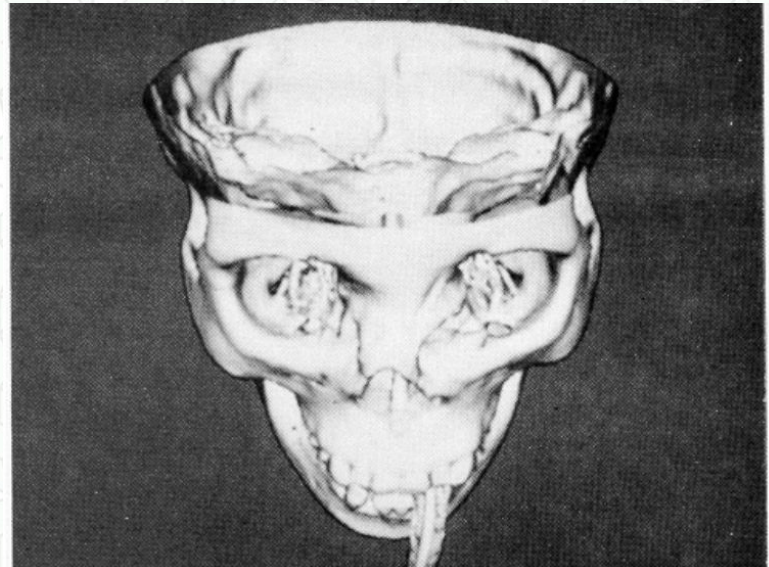
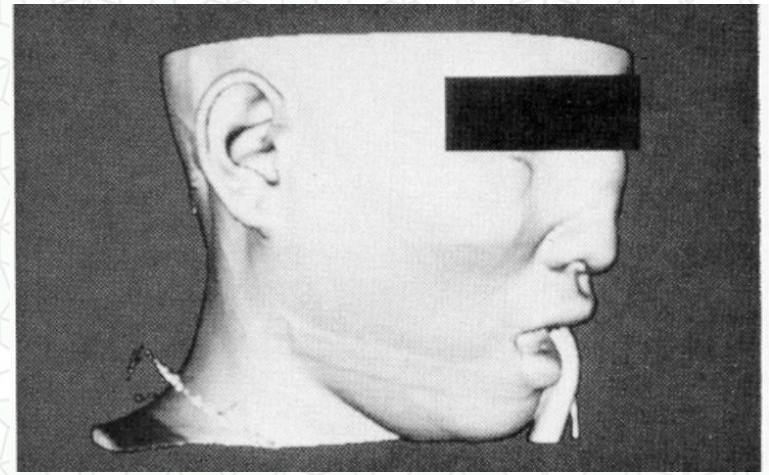
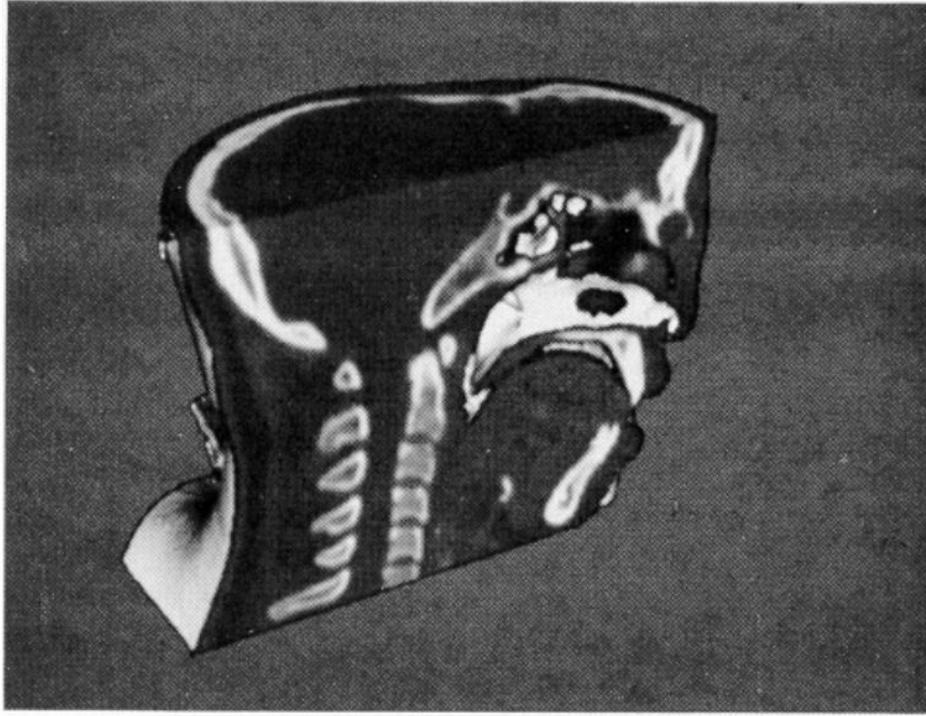


Marching Cubes

- 256 possible inside/outside labelings of each grid cube.
- Merging rotations...
15 unique cases to implement

"Marching Cubes: A High Resolution 3D Surface Construction Algorithm",
Lorensen and Cline, SIGGRAPH '87.





"Marching Cubes: A High Resolution 3D Surface Construction Algorithm", Lorensen and Cline, SIGGRAPH '87.

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- **General Position, Floating Point Equality**
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

“Assuming General Position...”

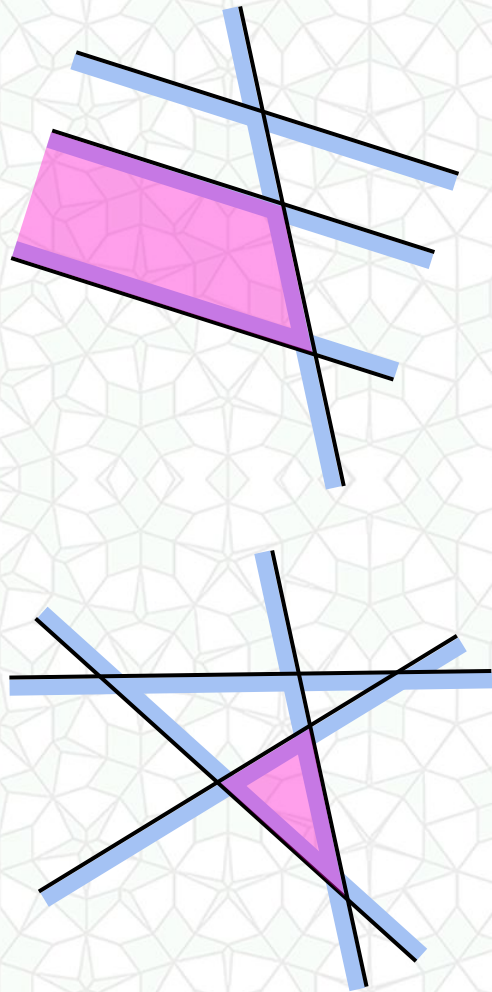
- Degeneracies in the input data cause problems
- To avoid problems in developing algorithms and to prove the correctness and performance of those algorithms,

We will often make assumptions, e.g.:

- No 2 points have the same x **and** same y coordinates
- No 3 points are collinear
- No 2 points lie on the same vertical line
- No 4 points lie on the same circle
- No 2 lines are parallel

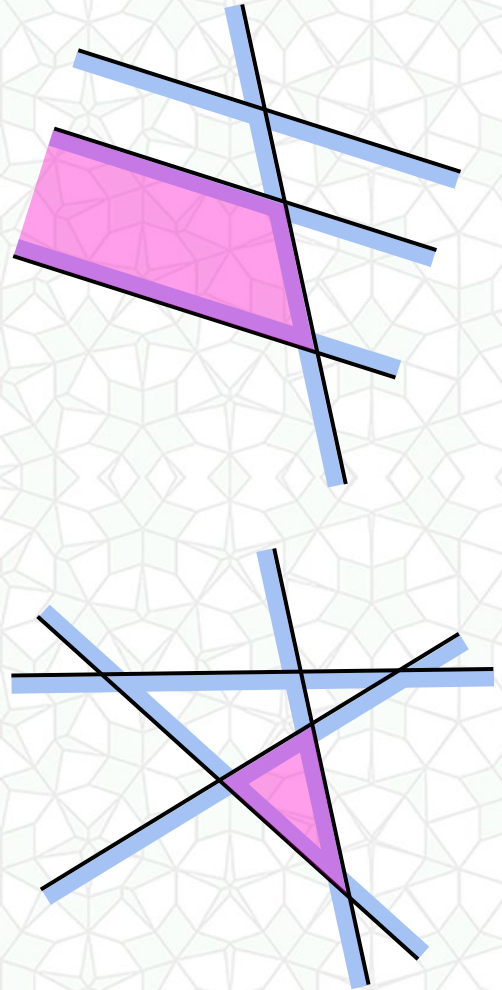
Homework 4 Redux

- *“A set of at least three half-planes with a non-empty intersection **such that not all bounding lines are parallel**”*
- Why put this clause in the problem statement?
 - Does this mean no 2 lines in the collection are parallel?
 - Or could all of the lines be parallel except one?
 - Is the region guaranteed to be bounded?



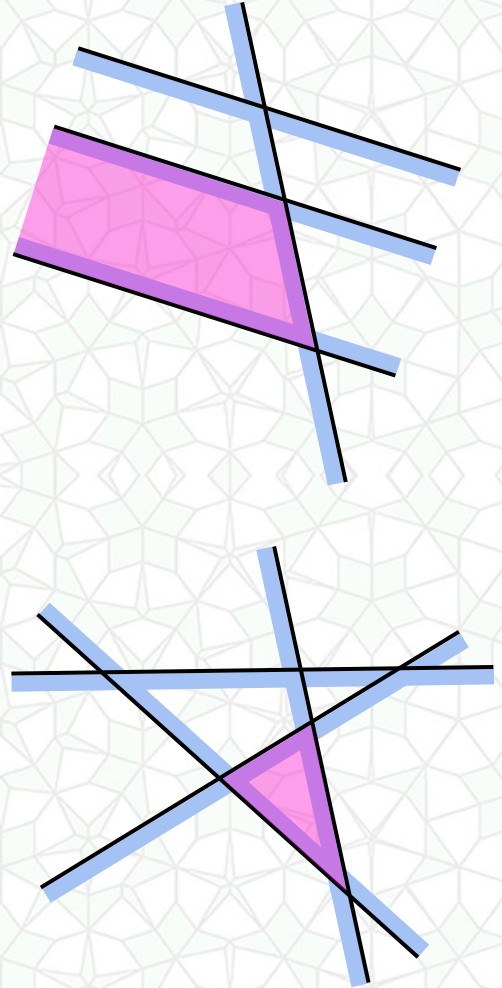
Homework 4 Redux

- *“A set of at least three half-planes with a non-empty intersection such that not all bounding lines are parallel”*
- Why put this clause in the problem statement?
 - Does this mean no 2 lines are in the collection are parallel? **No**
 - Or could all of the lines be parallel except one? **Yes**
 - Is the region guaranteed to be bounded? **No**



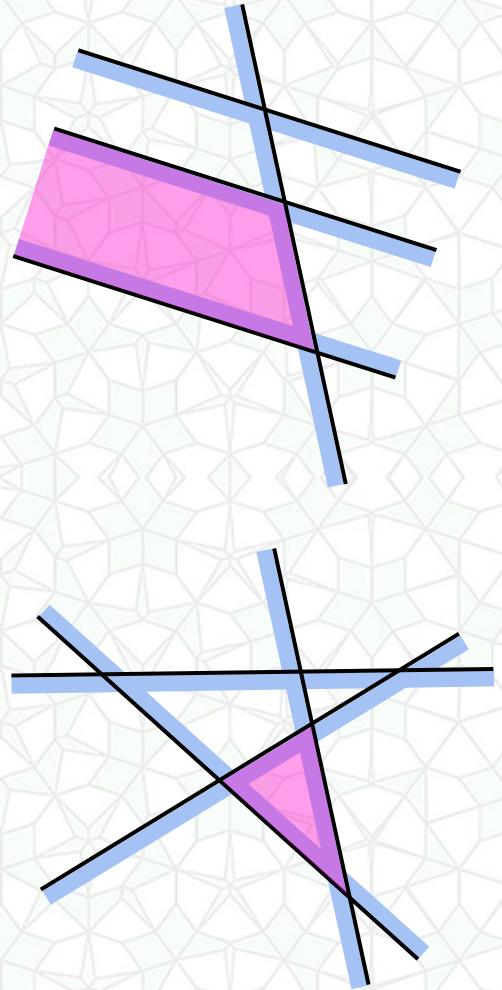
Homework 4 Redux

- *“A set of at least three half-planes with a non-empty intersection such that not all bounding lines are parallel”*
- How do we approach solving this problem?



Homework 4 Redux

- *“A set of at least three half-planes with a non-empty intersection such that not all bounding lines are parallel”*
- How do we approach solving this problem?
- *Recommendation:*
 - *Assume no lines are parallel (every line intersects every other line)*
 - *Assume only 2 lines intersect at a point*
- *Once you have the general problem solved/proved, then revisit the assumptions*



Robustness: Floating Point Equality

- Programming Mantra: Never compare two floats or doubles with ==
- Why not?

```
double a = 2/float(3);
```

```
double b = (5/3.0) * (7.0/5.0f) * (2/double(7));
```

```
assert (a == b);
```

Robustness: Floating Point Equality

- Programming Mantra: Never compare two floats or doubles with ==

- Why not?

```
double a = 2/float(3);
```

```
double b = (5/3.0) * (7.0/5.0f) * (2/double(7));
```

```
assert (a == b); ← this will probably fail!
```

$$\frac{\cancel{5}}{3} * \frac{\cancel{7}}{\cancel{5}} * \frac{2}{\cancel{7}} = \frac{2}{3}$$

- Even if we're more careful and use float & double consistently, the compiler is still free to use extra precision for the values of intermediate expressions.
- Optimized code might use registers for intermediate expressions (which often have higher precision than required by the type).

Common Robustness Workaround

- Instead compare to a tolerance or epsilon value, e.g.,

```
double a = 2/float(3);
```

```
double b = (5/3.0) * (7.0/5.0f) * (2/double(7));
```

```
assert (fabs(a-b) < 0.00001);
```

- But what is the right value for epsilon?

Common Robustness Workaround

- Instead compare to a tolerance or epsilon value, e.g.,

```
double a = 2/float(3);
```

```
double b = (5/3.0) * (7.0/5.0f) * (2/double(7));
```

```
assert (fabs(a-b) < 0.00001);
```

- But what is the right value for epsilon?

It depends on the application, data type, & overall scale of the data!

- epsilon way too big → *we risk computing the wrong answer*
- epsilon too small → *the original equality rounding error issue*
- What if roundoff error will accumulate or compound over time?

It will likely be impossible to appropriately set an epsilon!

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

Numerical Computing & Divide by Zero

- For numerical computing, divide by zero is the most common (is the only?) precision / rounding error that may cause a program to crash
- Otherwise, the program will always return a result
 - The result will be a good answer
 - It may be slightly off due to rounding error (the error is proportional to the types – e.g., float/double), but it is generally acceptable

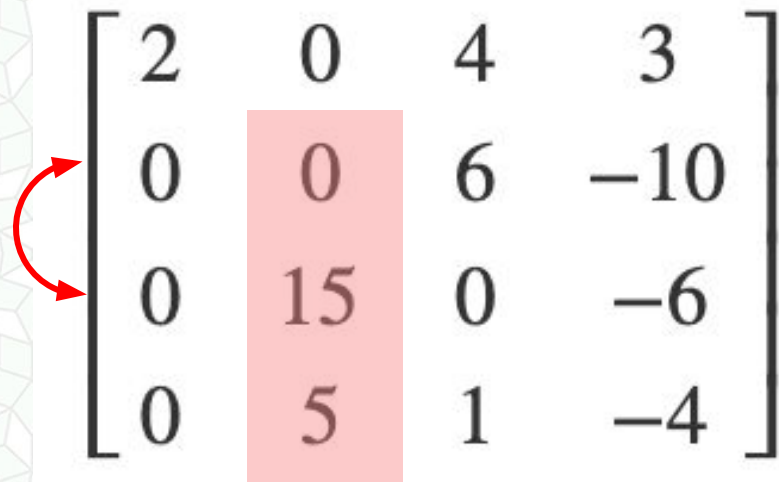
Factorization by Gaussian Elimination

- *When / in what course did you learn how to do this?*
- Multiply & subtract lower rows from the rows above
- We want to manipulate the matrix so that **all values in the lower triangle are zero.**
- **The diagonal should be non-zero**

$$\begin{bmatrix} U_{11} & U_{12} & U_{13} & U_{14} \\ 0 & U_{22} & U_{23} & U_{24} \\ 0 & 0 & U_{33} & U_{34} \\ 0 & 0 & 0 & U_{44} \end{bmatrix} \mathbf{x} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{bmatrix}$$

Factorization by Gaussian Elimination

- A pivot or row swap is necessary if the value in the target position is zero and would lead to a divide-by-zero when we try to compute the row multiplier necessary to produce zeros in that column in the lower rows.


$$\begin{bmatrix} 2 & 0 & 4 & 3 \\ 0 & 0 & 6 & -10 \\ 0 & 15 & 0 & -6 \\ 0 & 5 & 1 & -4 \end{bmatrix}$$

Factorization by Gaussian Elimination

- Divide by zero is not the only concern...
- We should also avoid division by very small values, e.g., epsilon:

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} = \begin{bmatrix} -\epsilon & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 - \epsilon \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} -\epsilon & 1 & 1 - \epsilon \\ 0 & -1 + \epsilon^{-1} & \epsilon^{-1} - 1 \end{bmatrix} \Rightarrow \begin{array}{l} x_2 = 1 \\ x_1 = \frac{(1 - \epsilon) - 1}{-\epsilon} \end{array}$$

*Correct answer: $x_1 = 1$
But we will have
robustness problems
if ϵ is very small!*

Factorization by Gaussian Elimination

- Divide by zero is not the only concern...
- We should also avoid division by very small values, e.g., epsilon:

$$\mathbf{Ax} = \mathbf{b}$$

$$\mathbf{A} = \begin{bmatrix} -\epsilon & 1 \\ 1 & -1 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} 1 - \epsilon \\ 0 \end{bmatrix},$$

$$\begin{bmatrix} -\epsilon & 1 & 1 - \epsilon \\ 0 & -1 + \epsilon^{-1} & \epsilon^{-1} - 1 \end{bmatrix} \Rightarrow$$

$$\begin{aligned} x_2 &= 1 \\ x_1 &= \frac{(1 - \epsilon) - 1}{-\epsilon} \end{aligned}$$

*Correct answer: $x_1 = 1$
But we will have
robustness problems
if ϵ is very small!*

*It's better to pivot / swap
rows for the row with the
largest value in this column*

Numerical vs. Combinatorial

- Use of a tolerance or epsilon is an appropriate approach for *numerical computing* (e.g. solving linear systems), where answers being slightly off is acceptable.
- However in geometry, the goal is not to compute numbers but rather structures (convex hull, Delaunay triangulation, etc).
- It is a *combinatorial problem*, not a numerical problem.

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- **Floating Point Bugs in Computational Geometry**
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

Ramshaw's Braided Lines

- Consider 2 lines,

$$l_1 : y = 9833x/9454$$

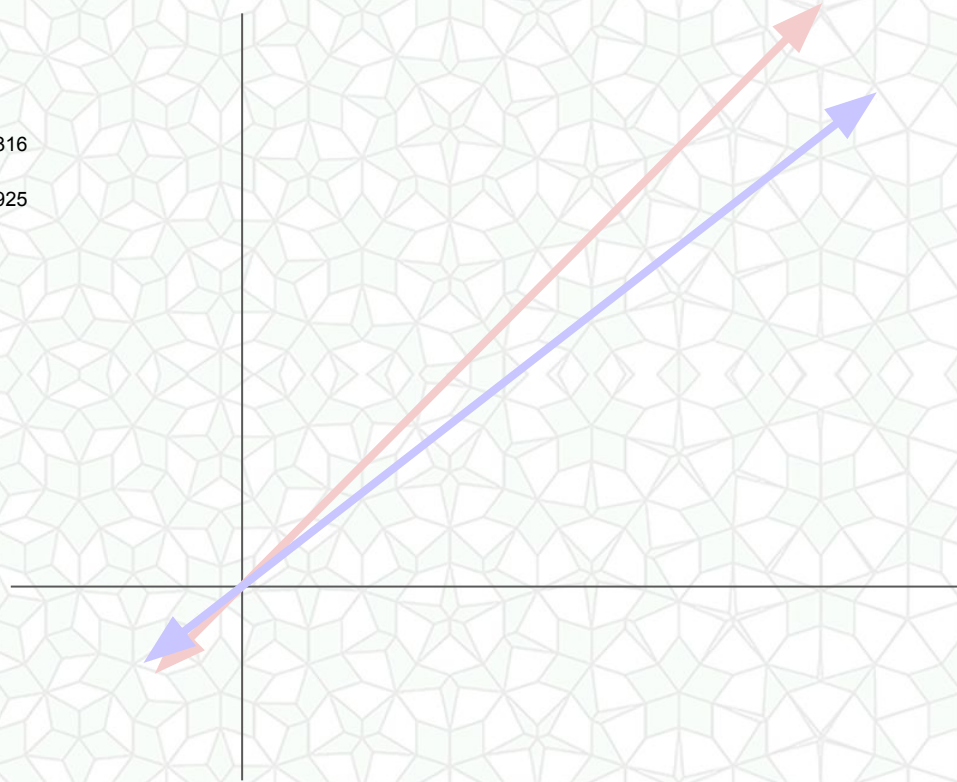
1.0400888512798816

vs.

$$l_2 : y = 9366x/9005$$

1.0400888395335925

both pass through the origin,
slope of l_1 is slightly larger than l_2



Ramshaw's Braided Lines

- Consider 2 lines,

$$l_1: y = 9833x/9454 \quad 1.0400888512798816$$

vs.

$$l_2: y = 9366x/9005 \quad 1.0400888395335925$$

both pass through the origin,

slope of l_1 is slightly larger than l_2

- This program computes and compares the y-value for each line at multiples of 0.001 between 0 and 1*

```
#include <iostream.h>
```

```
int main()
{
    cout.precision(12);
    float delta=0.001f;
    int last_comp=-1;
    float a=9833,b=9454,c=9366,d=9005;

    float x;
    for (x=0;x<0.1;x=x+delta) {
        float y1=a*x/b;
        float y2=c*x/d;

        int comp;
        if (y1<y2) comp=-1;
        else if (y1==y2) comp=0;
        else comp=1;

        if (comp!=last_comp) {
            cout << endl << x << ": ";
            if (comp==-1) cout << "l1 is below l2";
            if (comp==0) cout << "l1 intersects l2";
            else cout << "l1 is above l2";
        }

        last_comp=comp;
    }

    cout << endl << endl;
    return 0;
}
```

Ramshaw's Braided Lines

- Consider 2 lines,

$$l_1: y = 9833x/9454 \quad 1.0400888512798816$$

vs.

$$l_2: y = 9366x/9005 \quad 1.0400888395335925$$

both pass through the origin,
slope of l_1 is slightly larger than l_2

- This program computes and compares the y-value for each line at multiples of 0.001 between 0 and 1
- The program outputs that l_1 and l_2 intersect 24 times !?!?***
- If we switch `float` \rightarrow `double`, it still prints 1 false intersection (not the origin)

```
#include <iostream.h>
```

```
int main()
{
    cout.precision(12);
    float delta=0.001f;
    int last_comp=-1;
    float a=9833,b=9454,c=9366,d=9005;

    float x;
    for (x=0;x<0.1;x=x+delta) {
        float y1=a*x/b;
        float y2=c*x/d;

        int comp;
        if (y1<y2) comp=-1;
        else if (y1==y2) comp=0;
        else comp=1;

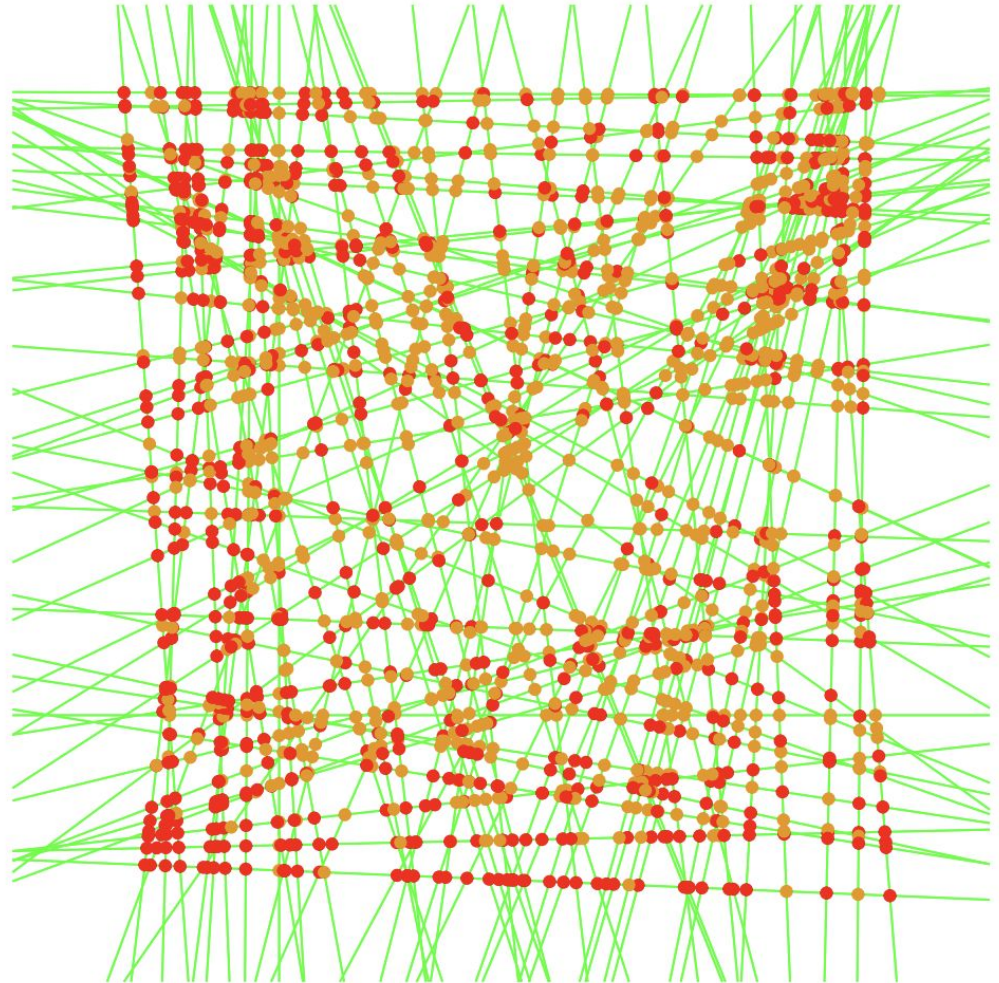
        if (comp!=last_comp) {
            cout << endl << x << ": ";
            if (comp==-1) cout << "l1 is below l2";
            if (comp==0) cout << "l1 intersects l2";
            else cout << "l1 is above l2";
        }

        last_comp=comp;
    }

    cout << endl << endl;
    return 0;
}
```


- Using floating point arithmetic:
- Take two random lines l_1 and l_2
- Compute intersection point p_{12}
- assert (point p_{12} lies on line l_1)
- assert (point p_{12} lies on line l_2)

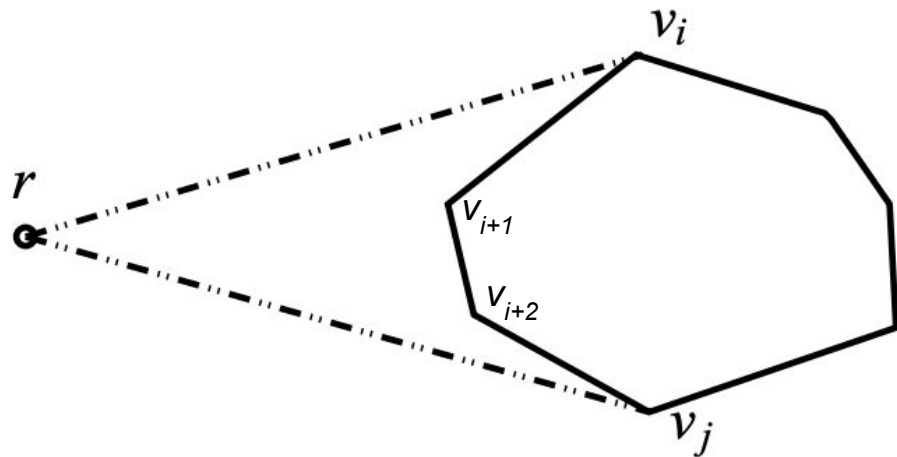
- Orange dots = 1 assertion fails
- Red dots = both assertion fails



Invited Lecture: “Real Numbers and Robustness in Computational Geometry”,
Real Numbers and Computers 2004,
Stefan Schirra

Incremental Convex Hull Construction

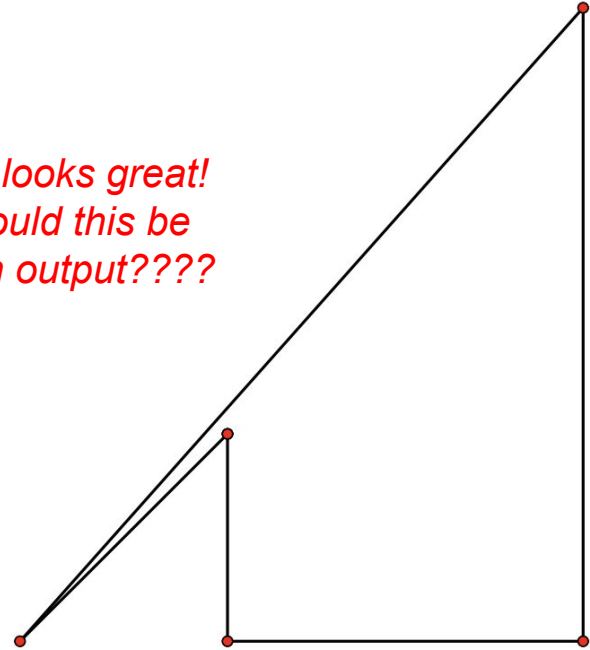
- Make a triangle with the first 3 points
- For each additional point r
 - Find an outside edge that is “visible” from r
 - Expand to a sequence of connected edges
 $v_i \rightarrow v_{i+1} \rightarrow v_{i+2} (\rightarrow \dots) \rightarrow v_j$
 - Remove middle points (e.g., v_{i+1} & v_{i+2}) from hull, add point r to hull



Incremental Convex Hull Construction

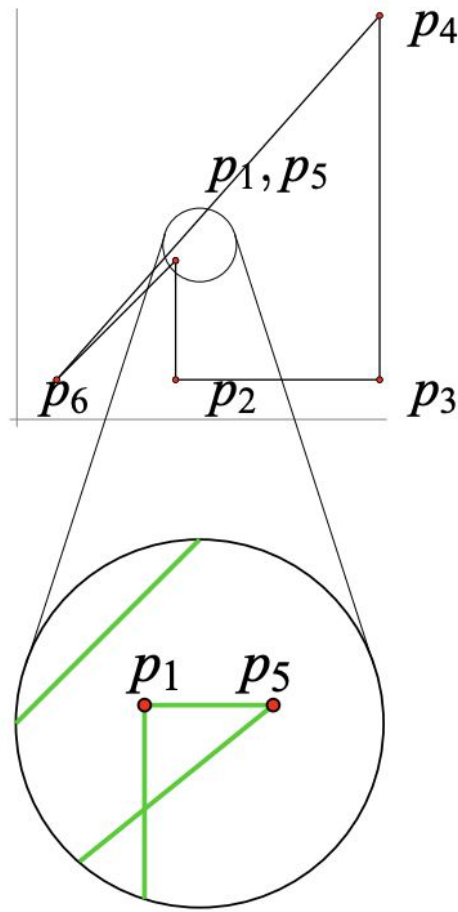
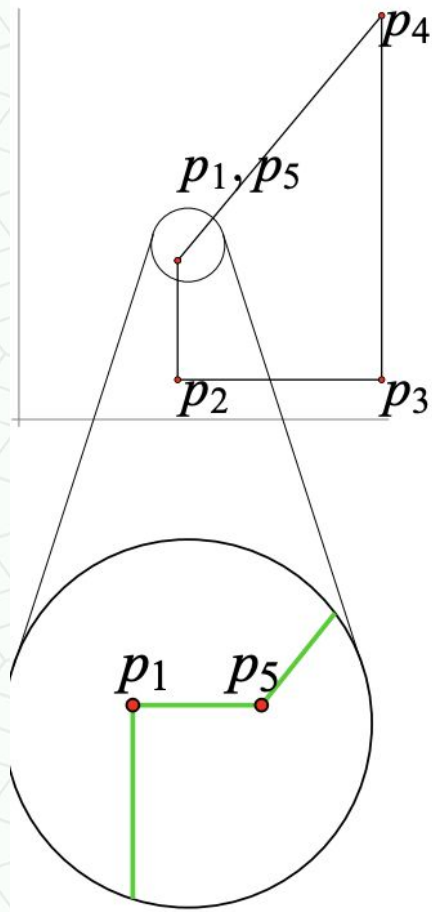
- Make a triangle with the first 3 points
- For each additional point r
 - Find an outside edge that is “visible” from r
 - Expand to a sequence of connected edges
 $v_i \rightarrow v_{i+1} \rightarrow v_{i+2} (\rightarrow \dots) \rightarrow v_j$
 - Remove middle points (e.g., v_{i+1} & v_{i+2}) from hull, add point r to hull

*Algorithm looks great!
So how could this be
a program output????*



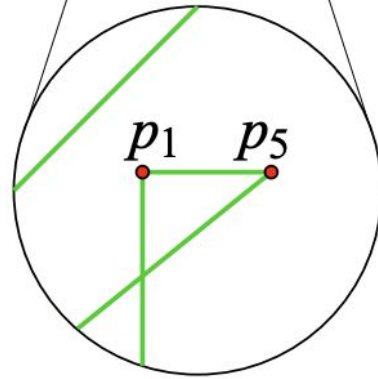
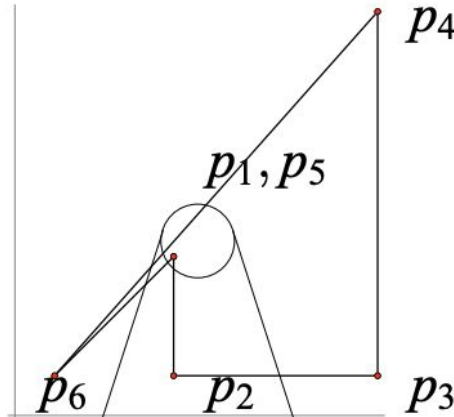
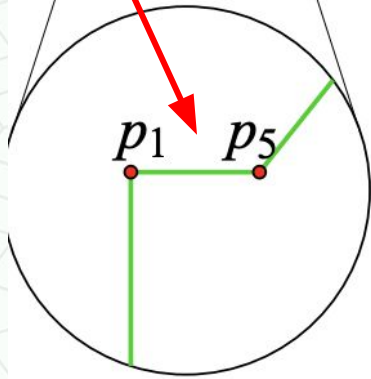
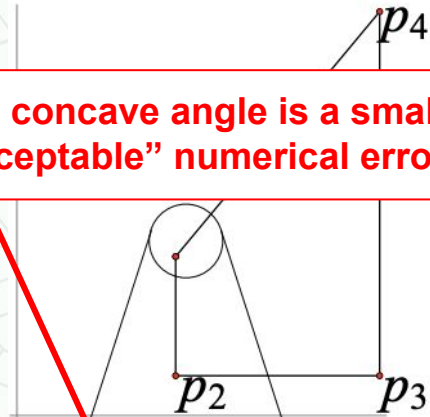
“Geometric Computing: The Science of Making Geometric Algorithms Work”, Kurt Mehlhorn

<https://people.mpi-inf.mpg.de/~mehlhorn/ftp/SoCG09.pdf>



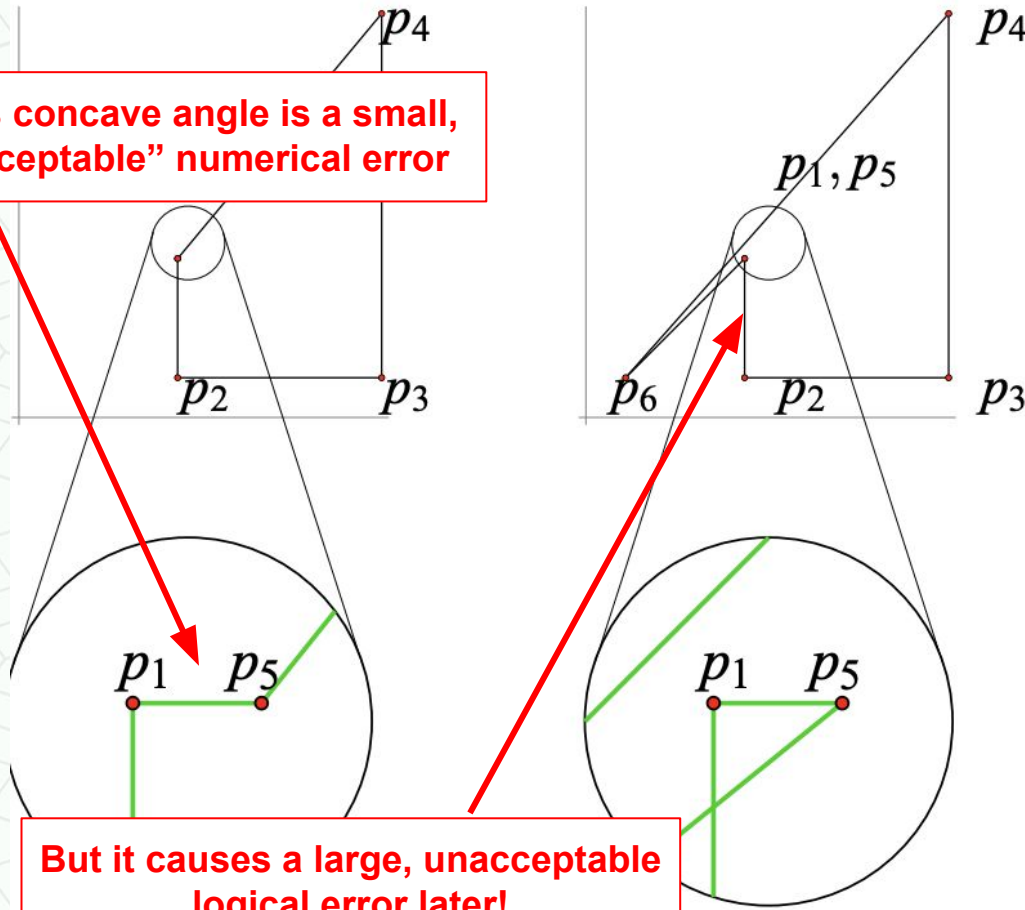
- the hull of p_1 to p_4 is correctly computed
- p_5 lies close to p_1 , lies inside the hull of the first four points, but float-sees the edge (p_1, p_4) . Concave corner at p_5 .
- point p_6 sees the edges (p_1, p_2) and (p_4, p_5) , but does **not** see the edge (p_5, p_1) .
- we obtain ...

This concave angle is a small, "acceptable" numerical error



- the hull of p_1 to p_4 is correctly computed
- p_5 lies close to p_1 , lies inside the hull of the first four points, but float-sees the edge (p_1, p_4) . Concave corner at p_5 .
- point p_6 sees the edges (p_1, p_2) and (p_4, p_5) , but does **not** see the edge (p_5, p_1) .
- we obtain ...

This concave angle is a small, "acceptable" numerical error



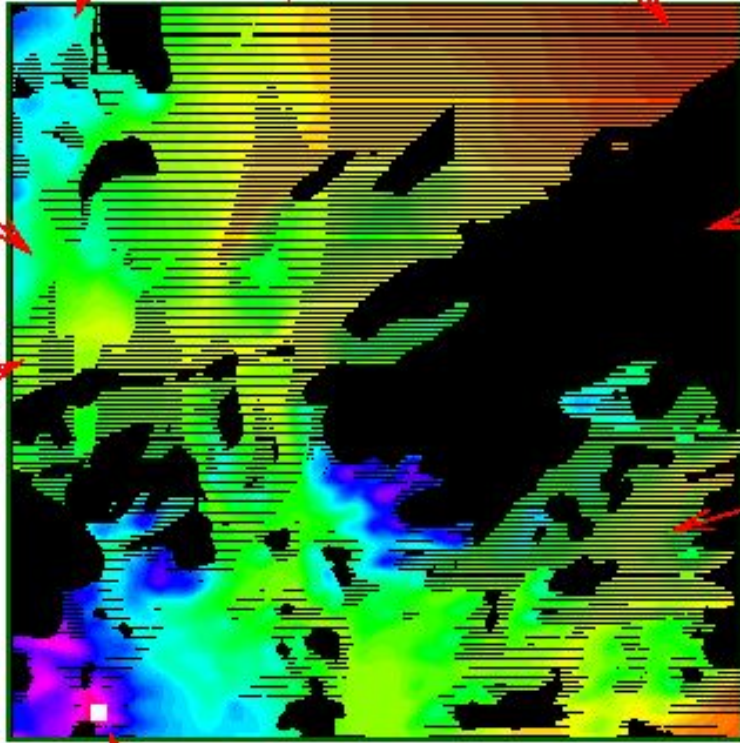
But it causes a large, unacceptable logical error later!

- the hull of p_1 to p_4 is correctly computed
- p_5 lies close to p_1 , lies inside the hull of the first four points, but float-sees the edge (p_1, p_4) . Concave corner at p_5 .
- point p_6 sees the edges (p_1, p_2) and (p_4, p_5) , but does **not** see the edge (p_5, p_1) .
- we obtain ...

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- **Floating Point Bugs in Computer Graphics**
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

Hue indicates elevation



Visible

Hidden

**Possibly
hidden**

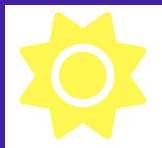
**Probably
hidden**

Observer

Possibly Hidden /
Probably Hidden:
If height is changed
by epsilon, the
visibility flips!

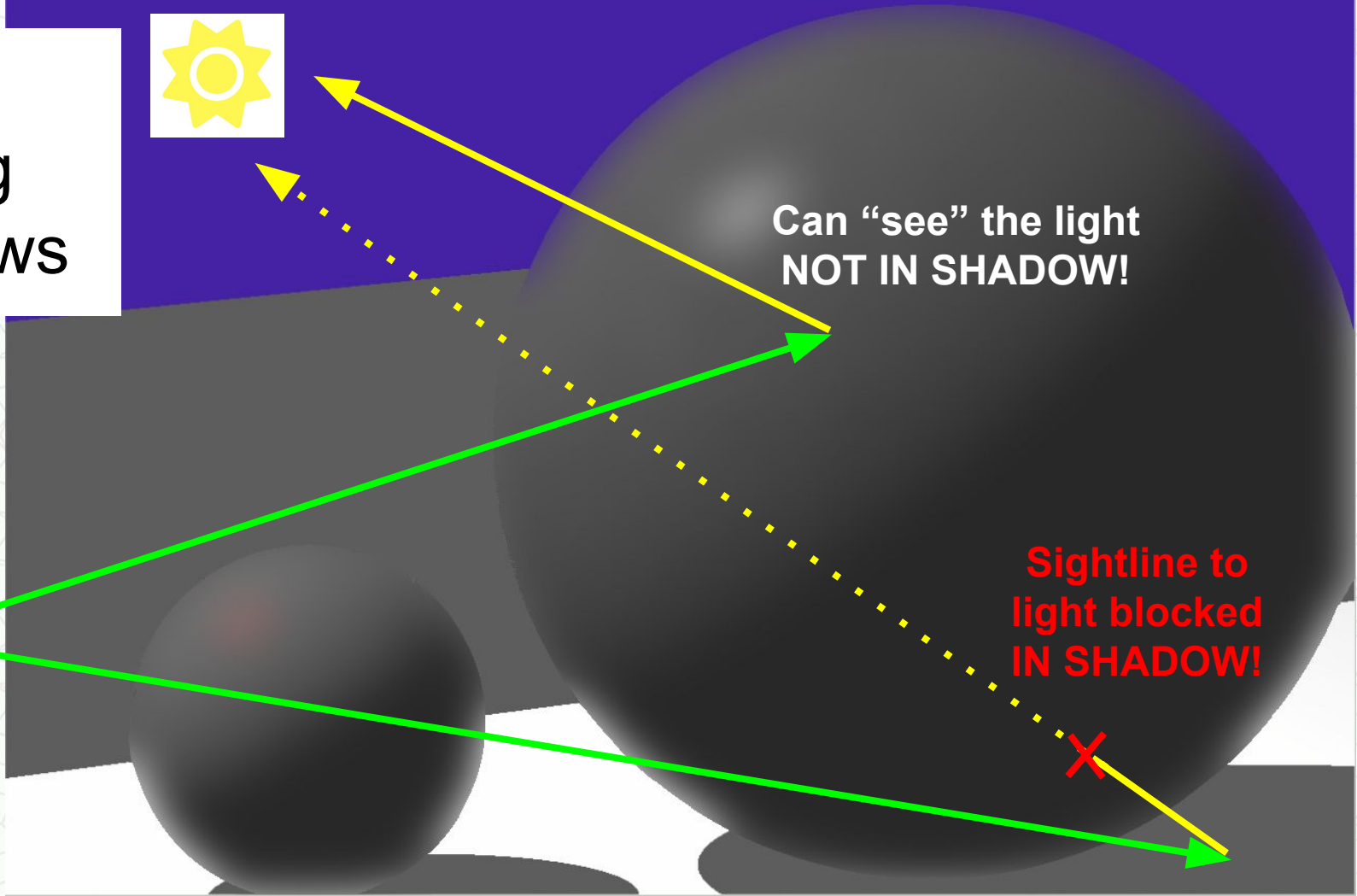
*The visibility
of one half
of the points
is uncertain!*

Ray Tracing Shadows

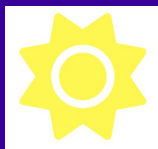


Can "see" the light
NOT IN SHADOW!

Sightline to
light blocked
IN SHADOW!



Ray Tracing Shadows



intersects light
@ t = 25.2

intersects sphere
@ t = -0.01

intersects sphere
@ t = 10.6

intersects light
@ t = 26.9

intersects sphere
@ t = 0.01

intersects sphere
@ t = 14.3

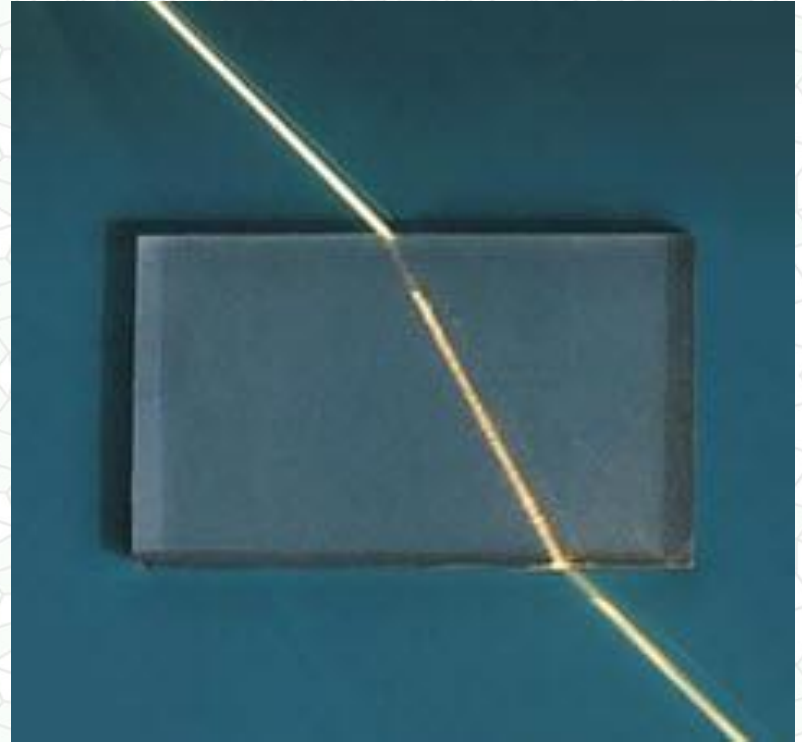
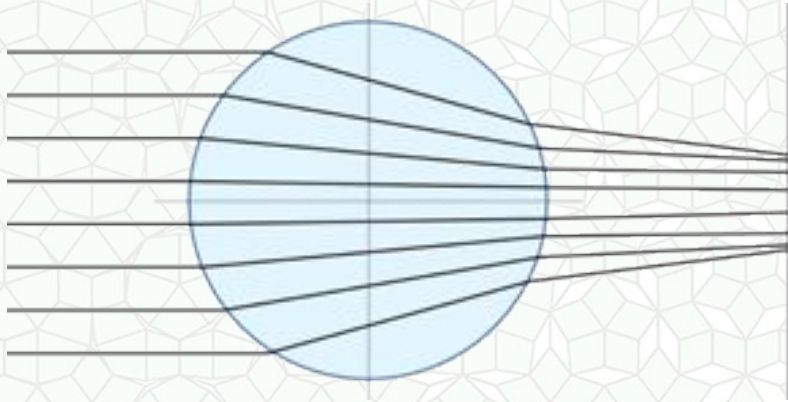


so we add epsilon
into our ray tracing

Image from
Zachary Lynn

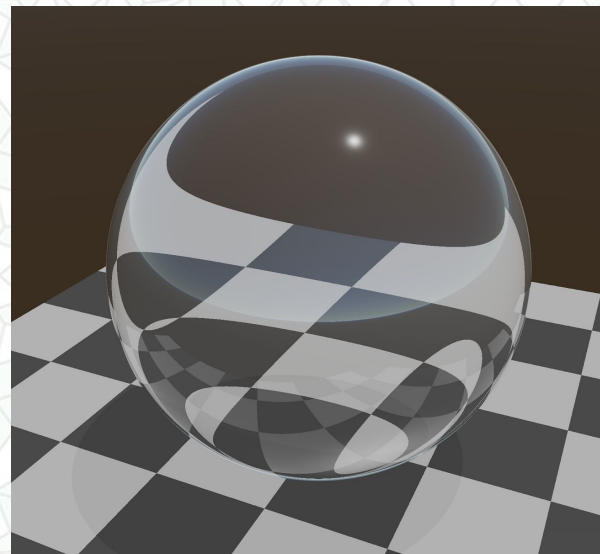
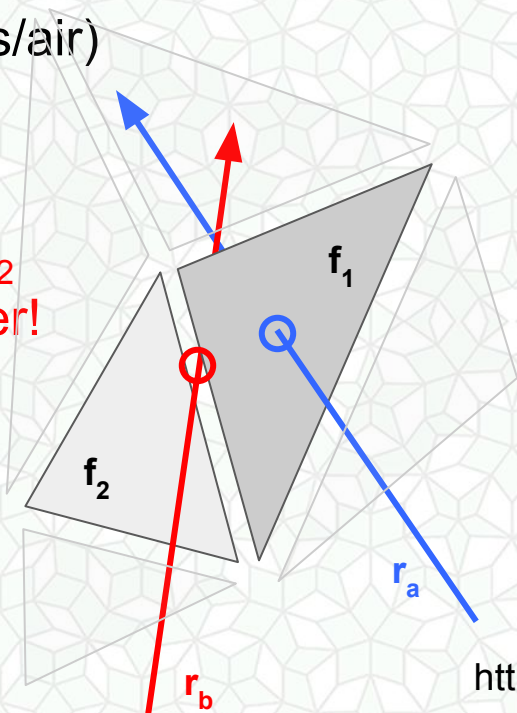
Ray Tracing Refraction

- To correctly simulate light as it bends/refracts through a medium denser than air (e.g., glass), we must know when a ray enters and when a ray exits an object.



Manage Rounding Errors in Ray Tracing

- Track which material (glass/air) we are currently inside
 - r_a intersects f_1
 - r_b must intersect f_1 or f_2 but NOT both or neither!
 - We cannot miss or double count intersections!
- Requires conforming mesh, (no t-junctions) & carefully designed intersection math



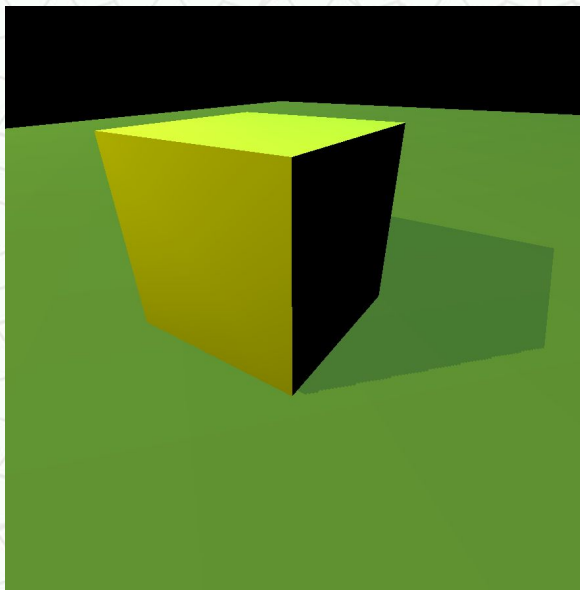
Jietong Chen

<https://cjt-jackton.github.io/RayTracing/>

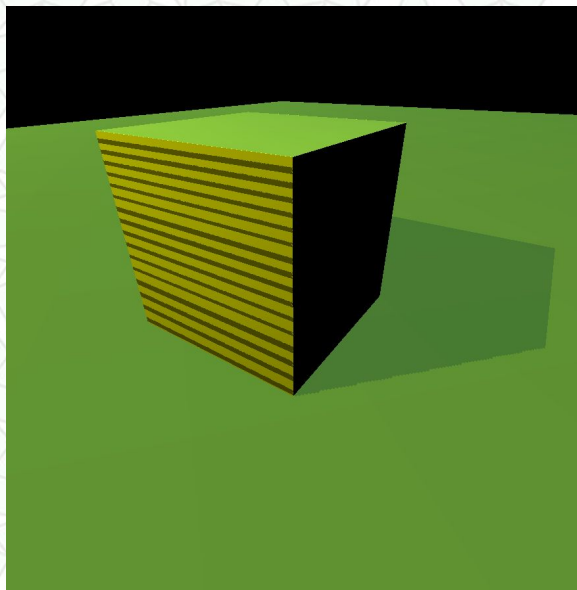
https://www.pbr-book.org/3ed-2018/Shapes/Managing_Rounding_Error

Epsilon a.k.a. Bias for **Shadow Maps**

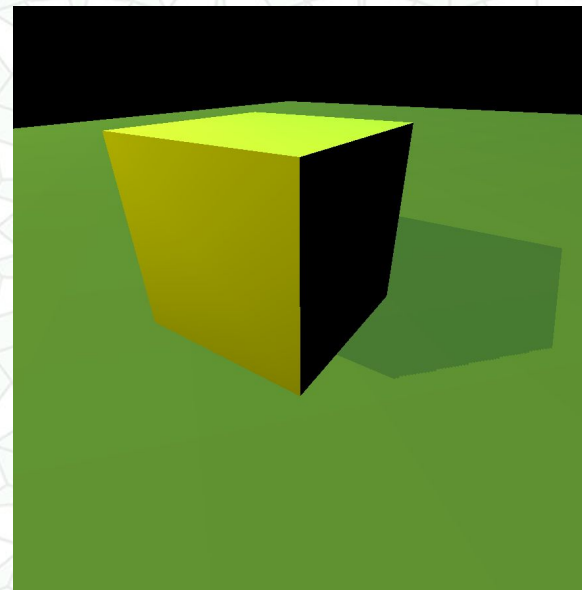
(GPU hardware-accelerated technique for shadows)



Correct image



Not enough bias



Way too much bias

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- **Real RAM vs. IEEE Floating Point**
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

Real RAM

- “A real RAM (random-access machine) is a mathematical model of a computer that can compute with exact real numbers instead of the binary fixed point or floating point numbers used by most actual computers.”
- Computers can only approximate a real RAM using floating point types.
- CGAL (The Computational Geometry Algorithms Library) and LEDA (A Library of Efficient Data Types and Algorithms) provide tools that allow us to write programs that work like they were running on a real RAM.

LEDA Data Types *proprietary but free for research*

Beyond standard C++ types, LEDA provides:

- **integer** - eliminates overflow by using unbounded memory for large numbers
- **rational** - mathematical definition of rational as the quotient of two **integers**.
- **bigfloat** - allows mantissa to be arbitrary level of precision instead of following the IEEE standard
- **real** - can compute the sign of a radical expression

Also has error checking, validation, proof of correctness algorithms

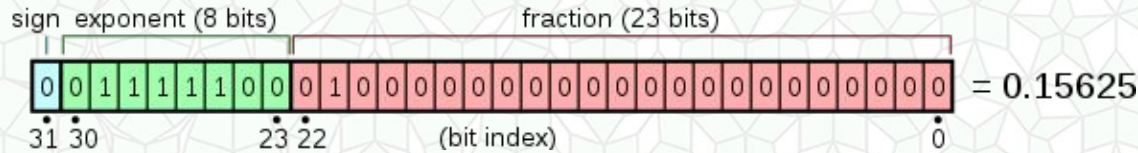
Boost (beyond C++ STL) *open source software!*

- The Multiprecision Library provides C++ types for:
 - integer,
 - rational,
 - floating-point, and
 - complex numbers.
- Precision may be:
 - arbitrarily large (limited only by available memory),
 - fixed at compile time, or
 - variable controlled at run-time.

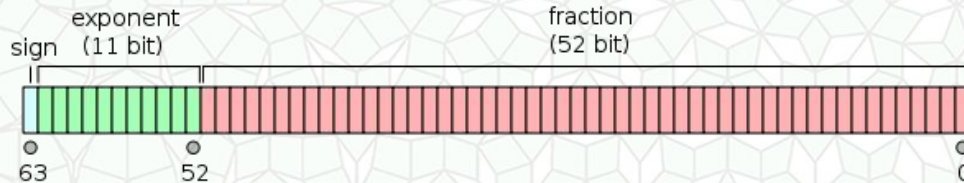
https://www.boost.org/doc/libs/1_83_0/libs/multiprecision/doc/html/boost_multiprecision/intro.html

The IEEE Floating Point Standard

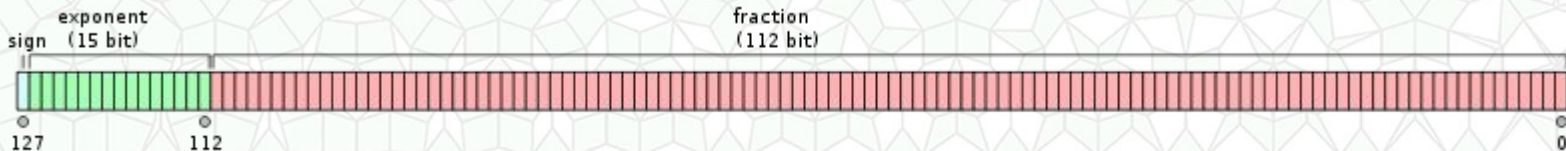
- IEEE *binary32* = C/C++ `float`



- IEEE *binary64* = C/C++ `double`

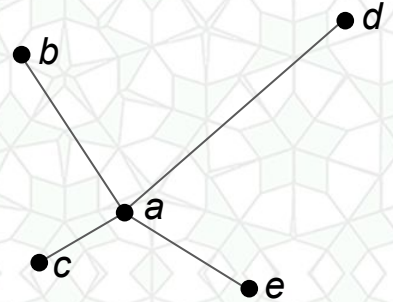


- IEEE *binary128* = [not (yet?) support by most hardware]



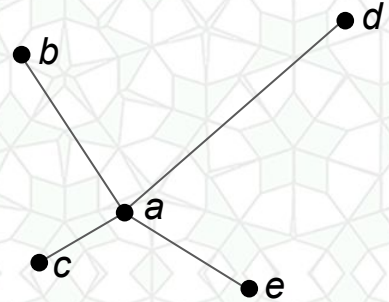
Programming Example with Irrational Numbers

- Problem: Given 5 points with integer coordinates, find the nearest neighbor to point a
- Compute the length of lines ab , ac , ad , ae
 - $\text{length}(ab) = \text{sqrt} ((x_a - x_b) * (x_a - x_b) + (y_a - y_b) * (y_a - y_b))$
 - *Note: the sqrt, will likely create irrational numbers!*
- Sort the lengths, return endpoint for shortest line length



Recommendation: *Avoid Creating Irrational Numbers*

- Problem: Given 5 points with integer coordinates, find the nearest neighbor to point a
- Compute the length of lines ab , ac , ad , ae
 - $\text{length}(ab) = \text{sqrt} ((x_a - x_b)^2 + (y_a - y_b)^2)$
 - *Note: the sqrt, will likely create irrational numbers!*
- Sort the lengths, return endpoint for shortest line length



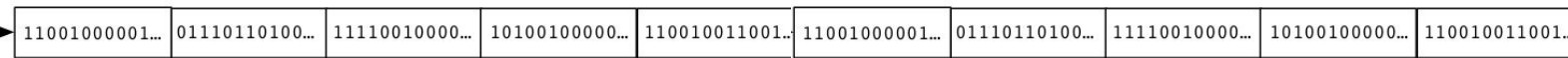
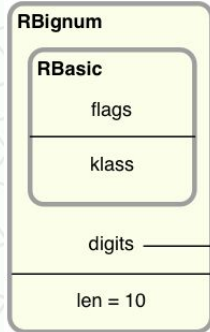
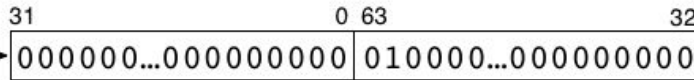
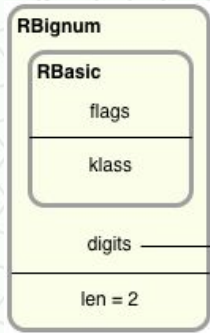
- Instead... compute & sort the squares of the line lengths
 - $\text{squared_length}(ab) = (x_a - x_b)^2 + (y_a - y_b)^2$
 - This is an integer! (because we have integer inputs)
- This will always return the correct answer to the original question.
WITHOUT creating irrational numbers!

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- **Arbitrary Arithmetic with Rational and Algebraic Numbers**
- Symbolic Computation, Floating Point Filters, Interval Computation
- Next Time: Binary Space Partition

Arbitrary Precision Arithmetic

- If we do not have irrational numbers in our program...
- We can store integers using a “BigNum” infinite precision integer type



- 64 bit binary integer = ~19 bit base 10 integer
- RSA Security requires at least 100 binary digits, but recommends 1000+ binary digits

Arbitrary Precision Arithmetic

- If we do not have irrational numbers in our program...
- We can store rational numbers as a ratio of two BigNums
- Reduce fractions whenever possible to minimize storage:

49578291287491495151508905425869578

74367436931237242727263358138804367

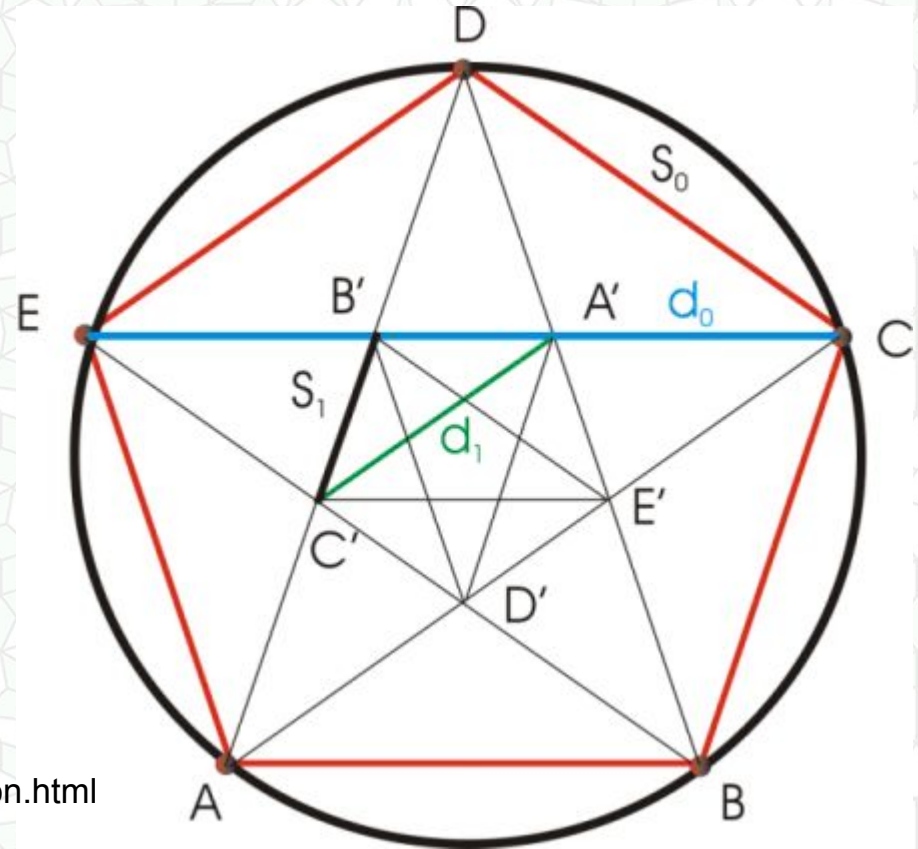


2

3

What if we *cannot* avoid Irrational Numbers?

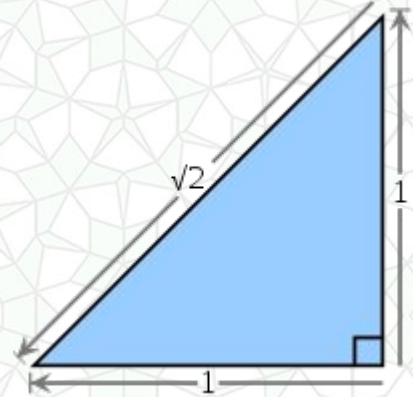
- *The dimensions of regular pentagon cannot be expressed with rational numbers!*
- Hippasos used a geometric analog of Euclid's algorithm to show that the ratio d_0/s_0 is an irrational number.



Algebraic Number

https://en.wikipedia.org/wiki/Algebraic_number

- A number that is a root of a non-zero polynomial in one variable with integer (or, equivalently, rational) coefficients.
- $\sqrt{2}$ is an algebraic number
- The golden ratio, $\phi = (1 + \sqrt{5}) / 2 \approx 1.61803$ is an algebraic number, it is a root of $x^2 - x - 1$
- All rational numbers are algebraic
- Some irrational numbers (e.g., $\sqrt{2}$ & ϕ) are algebraic
- Some irrational numbers are NOT algebraic
 - $\pi \approx 3.14159$ is not algebraic
 - $e = \sum_{n=0}^{\infty} \frac{1}{n!} = 1 + \frac{1}{1} + \frac{1}{1 \cdot 2} + \frac{1}{1 \cdot 2 \cdot 3} + \dots \approx 2.71828$ is not algebraic



Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- **Symbolic Computation, Floating Point Filters, Interval Computation**
- Next Time: Binary Space Partition

So, What's done in Practice?

- Input point coordinates are rational
- If we can limit to linear primitives – straight lines, not curves...
then the computations for most geometric problems will be rational, or at least algebraic.
- We can write software to implement & use basic arithmetic operations with all of the necessary types:
integers, big-nums, rational numbers, and even algebraic numbers
 - *And this is exactly what CGAL is doing with all of those C++ templates & typedefs!*
- Much of this can also be made to work with nonlinear primitives too!
 - *Avoid creating irrational numbers by working symbolically until output*

Improving Performance

- The challenge is efficiency.
 - CGAL: overhead for exact computation = 25% - 80% (depending on algorithm)
 - See also <https://www.cgal.org/exact.html>
- User is responsible for understanding exact vs. inexact computation
 - Writing good CGAL code (non-buggy, robust, accurate, and fast) takes skill
 - Leverage both non-exact and exact kernels in different places in same program
- Implementation of CGAL (& other libraries) is clever...
 - Don't use exact computation unless necessary
 - Work with floating point approximations most of the time
 - "Floating point filters": Automatically switch from a floating point representation to exact computation when the numbers are close to a floating point tolerance.
 - Use symbolic / lazy adaptive evaluation to delay exact computation until and only if it is actually necessary

Alternative for Real Number Computation?

- Take imprecision into account when designing and proving the algorithms
- “Topology oriented implementation”
 - Program will always return an answer, even if all computations are replaced by random numbers
 - Never crashes because of inconsistencies
- Has been done for some problems & algorithms in Computational Geometry
- However, because most proofs rely on “assume general position” or small tricks like “rotate everything a tiny amount” to break ties
Most work in Computational Geometry would need to be redone!

Outline for Today

- Homework 7 & Final Project Proposal Questions
- Last Time: Signed Distance & Level Sets
- General Position, Floating Point Equality
- Numerical Computing, Divide by Zero, & Gaussian Elimination
- Floating Point Bugs in Computational Geometry
- Floating Point Bugs in Computer Graphics
- Real RAM vs. IEEE Floating Point
- Arbitrary Arithmetic with Rational and Algebraic Numbers
- Symbolic Computation, Floating Point Filters, Interval Computation
- **Next Time: Binary Space Partition**

Next Time: Painter's Algorithm & Binary Space Partition



Bob Ross - "Peaceful Waters"

<https://www.twinchbrush.com/painting/peaceful-waters>

