Data Structures & Algorithms for Geometry

- Agenda:
  - Robustness of calculations
  - Prepare for final
  - Assignment #4

Fixed-point Representation

- Select an *n*-bit data size, partition *k*-bits for the integral part and *(n-k)*-bits for the fractional part
  - Numbers are evenly distributed
    - The difference between all representable numbers is always $1/(2^{n-k})$.
  - Color is usually done this way: 0-bits for the integral part, 8-bits for the fractional part
  - Also the way *all* real-time graphics math is done on processors without floating-point units
    - Pre-486, MIPS CPU in Playstation, Pre-68040, many “embedded” CPUs

Floating-point Representation

- Single precision:
  - 1-bit for sign
  - 8-bits for exponent
  - 23-bits for fractional part
- If $E \neq 0$, value = $(-1^{S} \times 1.F \times 2^{E-127})$
- If $E = 0$, value = $(-1^{S} \times 0.F \times 2^{-126})$
  - These are *denormalized* (denorm for short)

Magic Values

- IEEE-754 defines some magic values:
  - (E = 0, F = 0, S = 0) → 0
  - (E = 0, F = 0, S = 1) → -0
  - (E = 255, F = 0, S = 0) → $+\infty$
  - (E = 255, F = 0, S = 1) → $-\infty$
  - (E = 255, F $\neq$ 0) → Not a number (a.k.a., NaN)
    - More on NaN and Inf in a few slides

Distribution of Values

- As the exponent increases, the real difference between two values increases
  - (E = 200, F = 1) - (E = 200, F = 0) → $\sim 4 \times 10^{56}$
  - (E = 1, F = 1) - (E = 1, F = 0) → $\sim 1.4 \times 10^{-45}$
- Usually, this is okay.
  - This matches the notion of *significant digits*

NaN and Inf

- ±∞ are used to represent overflow cases
  - 1.0e20 / 1.0e-20 = +Inf
  - -1.0e20 / 1.0e-20 = -Inf
  - 1.0 / 0.0 = +Inf
- Not-a-number (NaN) is used to represent incalculable cases
  - 0 / 0 = NaN
  - √-1 = NaN
  - Inf - Inf = NaN

NaN Quirks

- **All** comparisons involving NaN are false.
  - Except NaN $\neq$ NaN, which is true.
- This means the following code segments will produce *different* results if `x` is NaN!

```
if (X < value) { a() } else { b() }
...
if (X >= value) { b() } else { a() }
```

NaN Quirks (cont.)

- NaN can *help* by avoiding the need for divide by zero tests
  - This code from the textbook works even if `Dot(p.n, ab)` is zero

```
int IntersectSegmentPlane(Point a, Point b, Plane p,
    float &t, Point &q)
{
    Vector ab = b - a;
    t = (p.d - Dot(p.n, a)) / Dot(p.n, ab);
    if (t >= 0.0f && t <= 1.0f) {
        q = a + t * ab;
        return 1;
    }
    return 0;
}
```

References

Hecker, Chris. 1996. Let's Get to the (Floating) Point. Game Developer Magazine. (Feb. / Mar. 1996). http://chrishecker.com/Miscellaneous_Technical_Articles#Floating_Point

Goldberg, D. 1991. What every computer scientist should know about floating-point arithmetic. ACM Computing Surveys. 23, 1 (Mar. 1991), 5-48. http://docs.sun.com/source/806-3568/ncg_goldberg.html

http://en.wikipedia.org/wiki/IEEE-754

Conversion and Representation Errors

- *Lots* of interesting, real numbers cannot be exactly represented
  - The more significant digits involved, the more *inexact* the representation will be
  - $\sqrt{2}$ get rounded to 1.41421356237309504880
    - This is an irrational number with infinite significant digits

Overflow and Underflow Errors

- If the result is too large (or too small) the result will overflow (or underflow)
  - Multiply two very large (or small) numbers
  - Divide a small number by a large number (or vice versa)
  - Overflows will result in $\pm$Inf
  - Underflows will result in $\pm$0

Round-off Errors

- Result of an operation has more significant digits than can be represented
  - X = (E = 3, F = 1.413351774) = 11.306814194
  - Y = (E = 7, F = 1.933333278) = 247.466659546
  - Results of X * Y:
    - True result: **2798.0595**3862742171622812747955322265625
    - Computer result: (E = 10, F = 2.732480049) = (E = 11, F = 1.366240025) = **2798.0595**703125

Digit-cancellation Errors

- Related to round-off errors
  - Subtracting nearly equal values
  - Adding or subtracting a large value and a small value
    - 1e20 + 1e-20 = 1e20
- Floating point arithmetic is *not* associative!
  - (9876543.0 + -9876547.0) + 3.45 = -0.5499999...
  - 9876543.0 + (-9876547.0 + 3.45) = -1.0

Input Errors

- The source data may have errors
  - Inexact measurement from a physical device
  - Errors from previous calculations
  - etc.

References

http://www.mpi-inf.mpg.de/~kettner/pub/nonrobust_ecgtr_04_a.html

http://en.wikipedia.org/wiki/Floating_point

Robust Comparisons

- Obviously, direct comparison for equality are just plain wrong

Robust Comparisons

- Obviously, direct comparison for equality are just plain wrong
- First improvement: compare absolute difference to some small value, ε

```
if (fabs(x - y) < epsilon) { ... }
```

  - Called *absolute tolerance*
  - Picking ε that works for a range of values is difficult or impossible
    - `sqrt(FLT_EPSILON)` is usually a good choice

Robust Comparisons (cont.)

- Second improvement: compare abs ratio to 1.0
  - Called *relative tolerance*

```
if (fabs((x / y) - 1.0) <= epsilon)
```

    - Assumes |x| < |y|

```
if (fabs((x - y) / y) <= epsilon)
```

    - First re-write

```
if (fabs(x - y) <= epsilon * fabs(y))
```

    - Eliminate division

```
if (fabs(x - y) <= epsilon * max(fabs(x),
  fabs(y)))
```

    - Eliminate |x| < |y| assumption

Robust Comparisons (cont.)

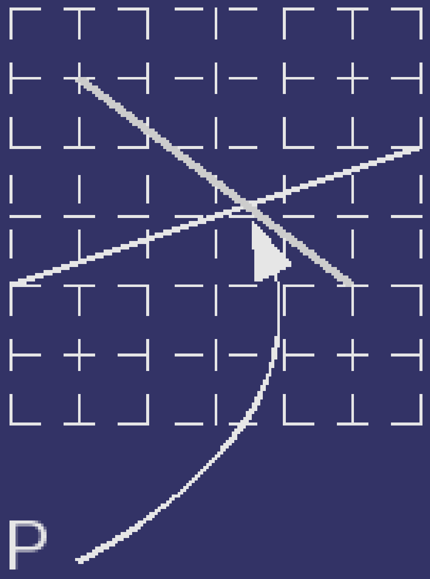
- Tolerances do have one ugly side-effect

Robust Comparisons (cont.)

- Tolerances do have one ugly side-effect
  - A = B and B = C does *not* imply A = C

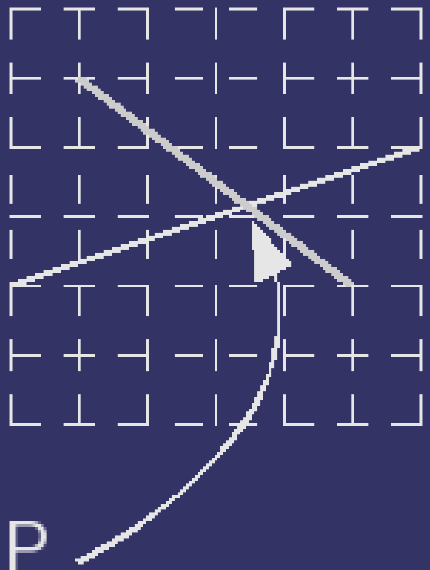
Thick Planes Revisited

- Not all numbers can be exactly represented in floating-point, so the true intersection point of a line and a plane may not be representable
  - Other object-object intersections similarly affected
  - Clipping the line to the plane effectively moves part of the line



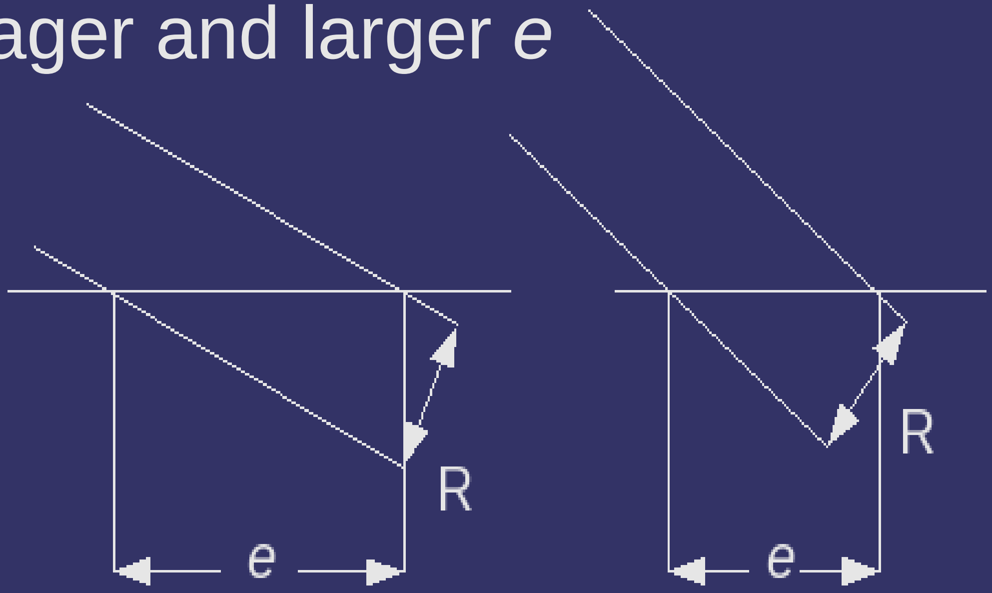
Thick Planes Revisited

- Let *e* be the maximum error in the calculated *P*, then the thick plane has radius *r*, *r* > *e*
  - How do we select an appropriate *e*?



Thick Planes Revisited

- Let *e* be the maximum error in the calculated *P*, then the thick plane has radius *r*, *r* > *e*
  - How do we select an appropriate *e*?
  - A the line and plane become more parallel, a selection of R leads to lager and larger *e*
  - The selection of R limits the size of line segments or polygons that we can track
    - Pick R based on the size of the smalled line or polygon



Next week...

- Final:
  - **Tuesday**, December 11th at 7:45PM.
  - DO NOT BE LATE!

Legal Statement

- This work represents the view of the authors and does not necessarily represent the view of IBM or the Art Institute of Portland.
- OpenGL is a trademark of Silicon Graphics, Inc. in the United States, other countries, or both.
- Khronos and OpenGL ES are trademarks of the Khronos Group.
- Other company, product, and service names may be trademarks or service marks of others.