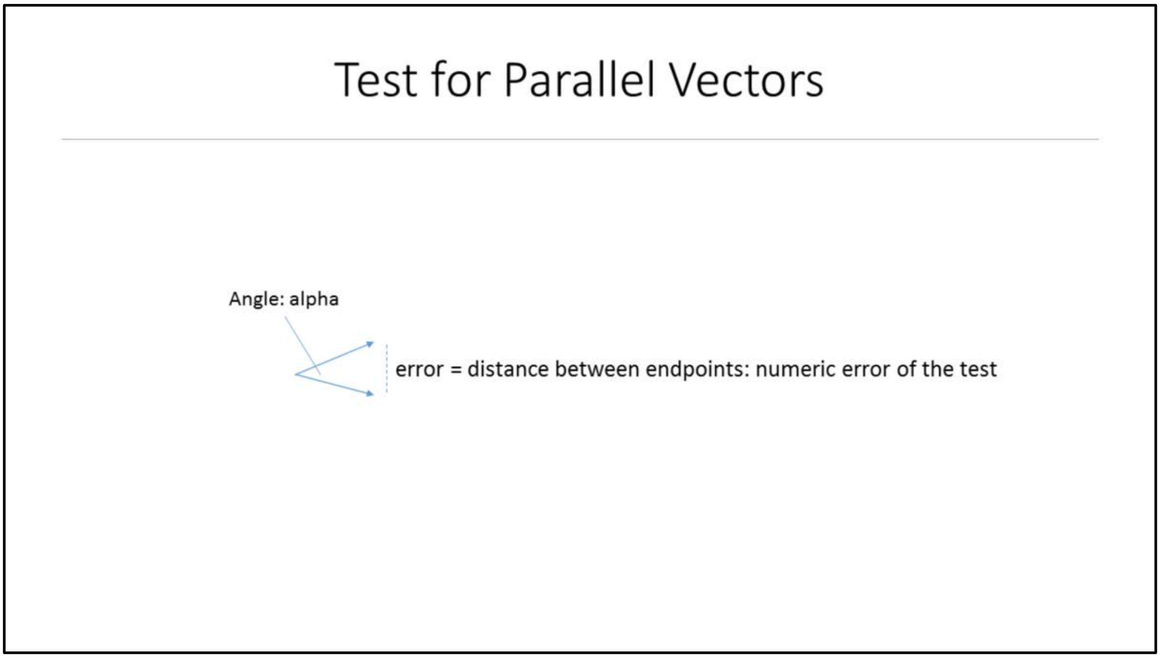
Two *Normalized* Vectors



- Given two parallel vectors we can compute cosine of the angle between them with the dot product
- This can give rise to a parallel vector test using the dot product
- We can define the error of how parallel vectors are by checking the distance between each vector’s endpoint when we center the vectors at the origin

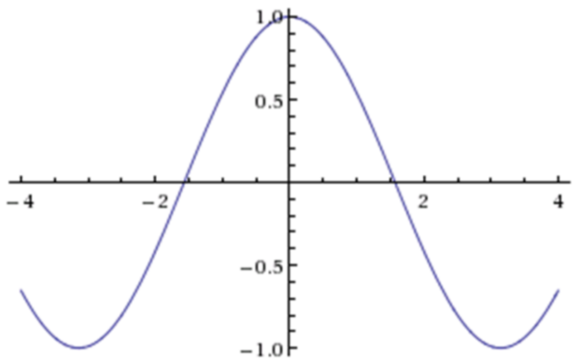
First Draft

```
bool Parallel( Vec3 a, Vec3 b )
{
        if ( abs( dot( a, b ) ) < epsilon )
                return true;

        return false;
}
```

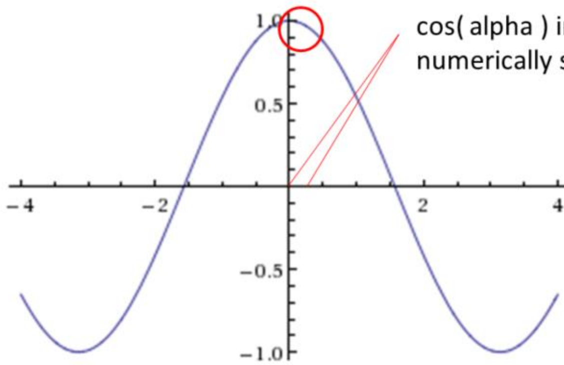
First Draft Problem

- Dot product computes cos alpha
  - For normalized vectors
- cos maps alpha to -1, 1
  - We computed abs(cos(alpha)), for a mapping of 0, 1
- Mapping is non-linear
- Granularity lessens near angles of zero

Nonlinearity of cos(alpha)



Nonlinearity of cos(alpha)



cos(alpha) in this range all have numerically similar results!

Overcoming First Draft Problems

- Compute acos(cos(alpha))
  - Computes actual angle, avoids non-linearity issues
  - Requires trig function acos
- Use alternative solution to avoid trig function

Second Draft

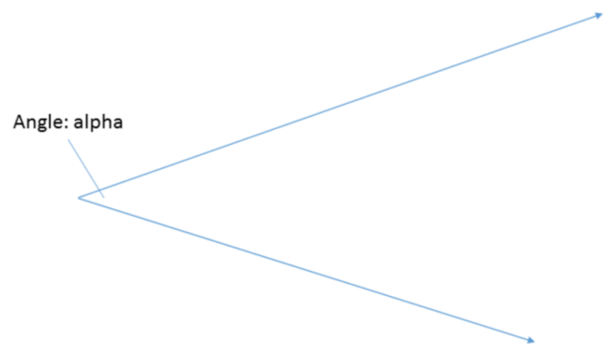
```
bool Parallel( Vec3 a, Vec3 b )
{
        if ( abs( a.x - b.x ) < epsilon &&
             abs( a.y - b.y ) < epsilon &&
             abs( a.z - b.z ) < epsilon )
        {
               return true;
        }

        return false;
}
```

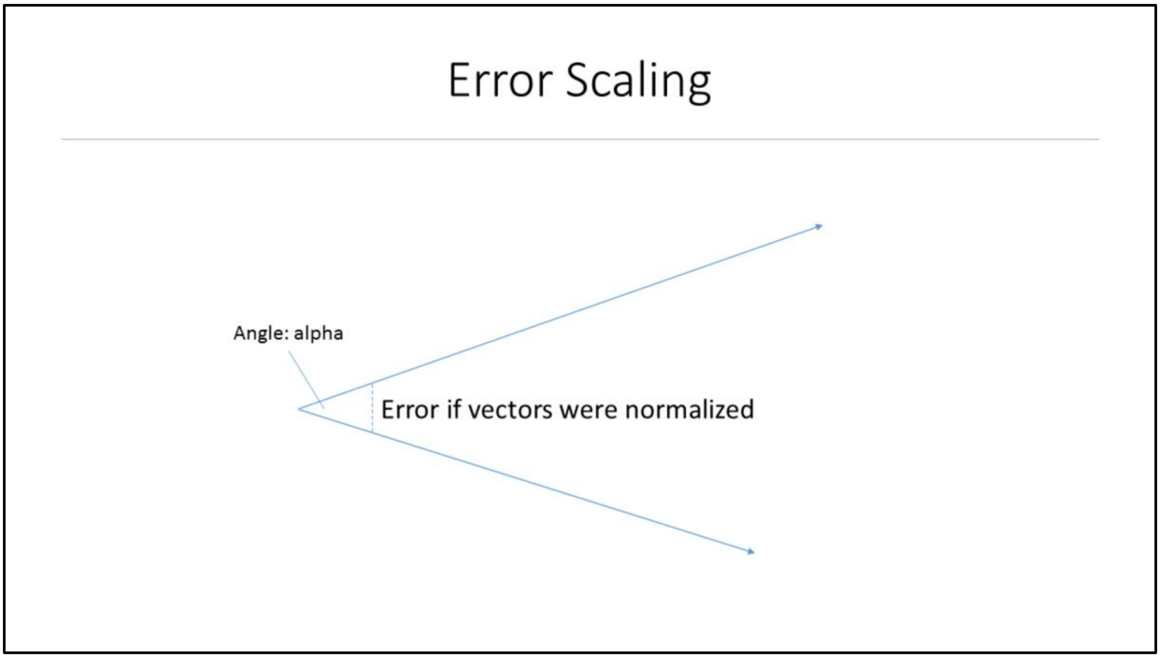
Second Draft Pros/Cons

- Pros:
  - Doesn’t have non-linearity of cos(alpha)
  - Doesn’t require trig function acos

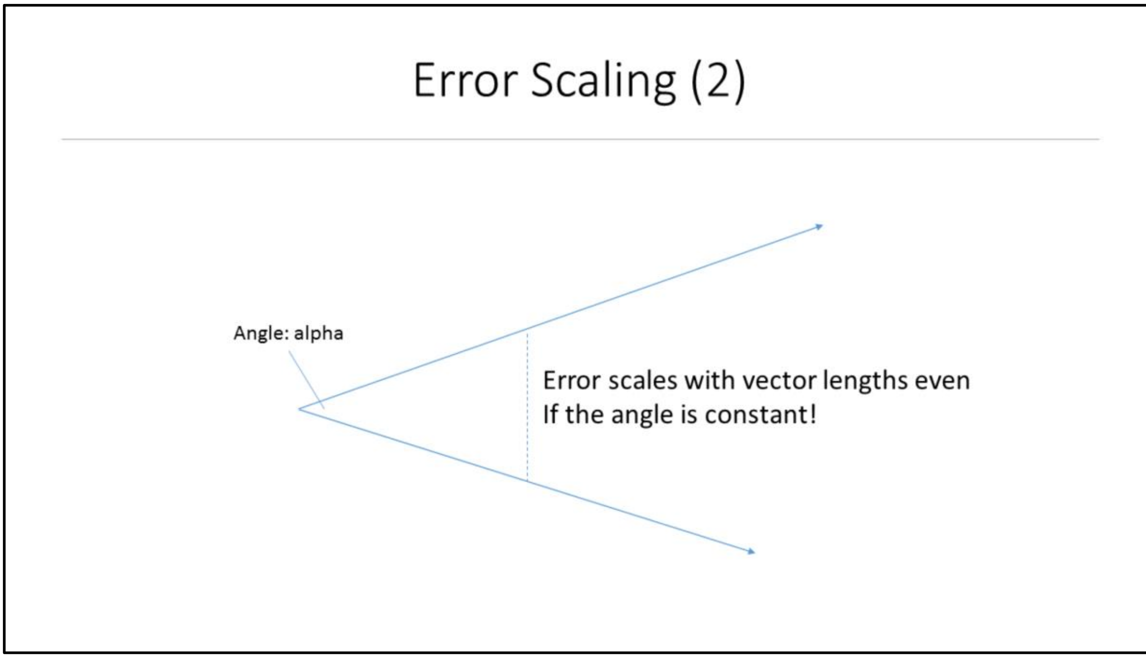
- Cons:
  - Only works on vectors of the same length

Two Non-Unit vectors





- Assume we detect that the angle between these two vectors is less than epsilon
- The distance error (as defined in the first slide) is small when vectors are short



- For longer vectors, even while alpha is constant and less than epsilon, the distance error grows

Error Scaling Causes Problems

- Imagine a 3D polyhedron
- Take the normals of two faces (ignore size of faces)
- Are they parallel?
  - This tests for coplanar faces

Error Scaling Causes Problems

- Imagine a 3D polyhedron
- Take the normals of two faces (ignore size of faces)
- Are they parallel?
  - This tests for coplanar faces

Side view of 2 faces and their normals on the surface of a polyhedron

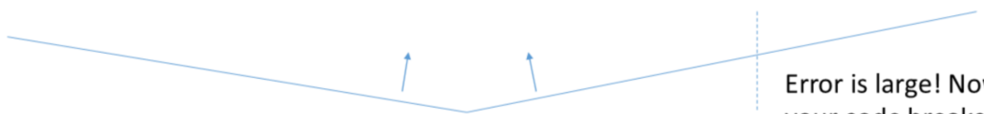
Error Scaling Causes Problems

- Imagine a 3D polyhedron
- Take the normals of two faces (ignore size of faces)
- Are they parallel?
  - This tests for coplanar faces

Error is small

Error Scaling Causes Problems

- Imagine a 3D polyhedron
- Take the normals of two faces (ignore size of faces)
- Are they parallel?
  - This tests for coplanar faces



Error is large! Now
your code breaks.

Conclusion?

- The coplanar example shows:
  - The scale of input vectors matters in some cases
  - Coplanarity tests are difficult

- Side note:
  - In practice a coplanarity test for faces on a polyhedron would usually involve one or two point to plane tests, and the normals themselves aren't parallel tested at all.

Ideal Test Requirements

- Takes angle between two vectors into consideration
  - First two tests achieved this
- Considers how error scales with vector lengths
  - This can give rise to slightly more robust results
- Implemented with relative epsilon comparison
  - Previous tests only used absolute epsilon
  - Relative is important since vector lengths may not be similar

- We would like a test that considers the scale of each vector somehow
- Longer vectors would require a different epsilon value than shorter vectors
- If the two vectors are of different lengths, the test must still be valid and compute appropriate epsilon values

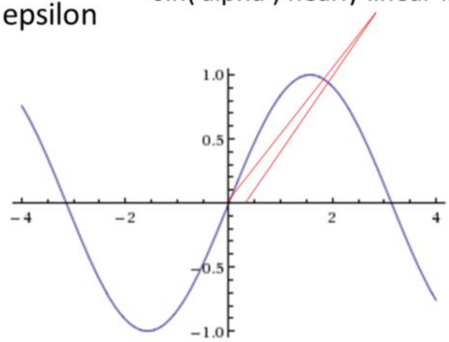
My Attempted Solution

```
bool Parallel( Vec3 a, Vec3 b )
{
	Vec3 c = Cross( a, b );
	float l = Length( c );

	if ( l < epsilon * sqrt( LengthSq( a ) * LengthSq( b ) ) )
		return true;

	return false;
}
```

Solution Explanation

- sin(alpha) nearly linear for small alphas
- Compute approximate relative epsilon
  - Scale of a and b are factors
- Perform relative epsilon test

sin(alpha) nearly linear in this range!



Recall: a x b = |a||b|sin(alpha) * n

Best Solution Yet

```
bool Parallel( Vec3 a, Vec3 b )
{
        float k = Length( a ) / Length( b );
        b *= k;

        if ( abs( a.x - b.x ) < epsilon &&
             abs( a.y - b.y ) < epsilon &&
             abs( a.z - b.z ) < epsilon )
        {
                return true;
        }

        return false;
}
```

Details

- Thanks to Christer for sharing this solution with me
- If vectors a and b are parallel then:
  - For some k, a = b * k
- Solve for k, scale b to a's scale
- Implement the 2^{nd} draft solution from previous slides

Pros and Cons

- Pros:
  - No non-linearity issues
  - Fast, only 2 square roots
- Cons:
  - I'm unsure of epsilon details for vectors with wildly different sized components
  - Check for division by zero
  - Should there be a bias to scale to the shorter or longer vector? Shorter might be better…

Potential Improvements

- Find the axis q where b the longest and non-zero
- Compute k as a.q / b.q
  - Avoid division by very small number
  - Avoid square root computation
- Try to pick a and b such that k <= 1 Allows multiplication with k in as numerically stable a way as possible

Have a Solution? Comments?

- Share it in the comments below, or email me!
- Resources:
  - Tolerances Revisited, Ericson
  - Essential Mathematics, Van Verth