# Geometry and Geometric Programming II

CMSC425.01 Spring 2019

Still at tables …

# Administrivia

- Project 1 submission
  - Name as follows:     Lastname-Firstname.zip.
  - For example, for TA Flores that would be Flores-Alejandro.zip
  - From the project folder, delete all folders except for Assets and ProjectSettings.
  - Library, Packages, Logs, and Temp are not necessary.

- Lectures online
  - Working to improve them – better audio, better handwriting
  - Get them up faster

- Looking for additional readings
  - http://www.hiteshpatel.co.in/ebook/cg/Computer_Graphics_C_Version.pdf
  - https://nccastaff.bournemouth.ac.uk/jmacey/CGF/slides/Lecture6VectorsAndMatrices4up.pdf

# Today’s question

# Computing distances, directions, orientations

# 425 != 427

- We will do considerable math from 427, but not all

Objectives in 425:

- Solve some problems important in game design in particular

- Introduce you to graphics math thinking so you can pick on your own

# Review from last class. Questions?

- After today you should be able to use:

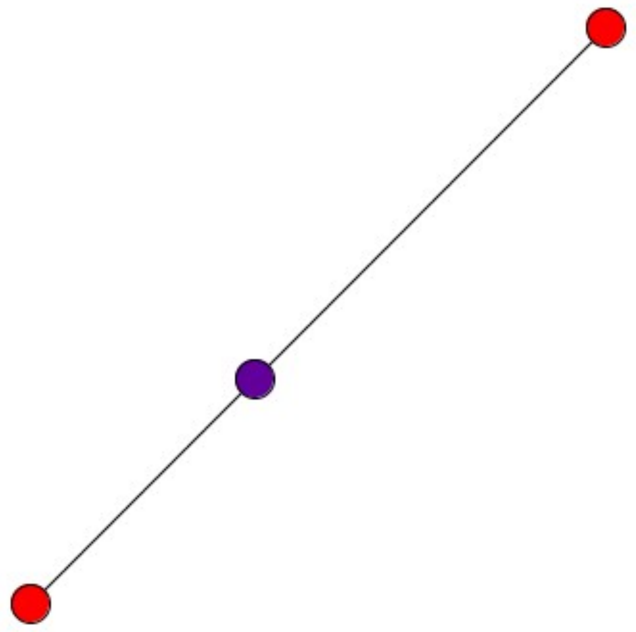
1) Affine data types and operations

   Vector addition, point subtraction, point-vector additions, etc.

2) Affine/convex combinations
3) Euclidean
   1) Dot/inner product
   2) Length, normalization, distance, angle, orthogonality
4) Orthogonal projection
5) Doing it in Unity

# Review: point-vector line $r = p + tv$

- Line between p (100,400) and q (400,100)
- (y inverted, 0 at top)
- Parametric in t
- Formula in this case?



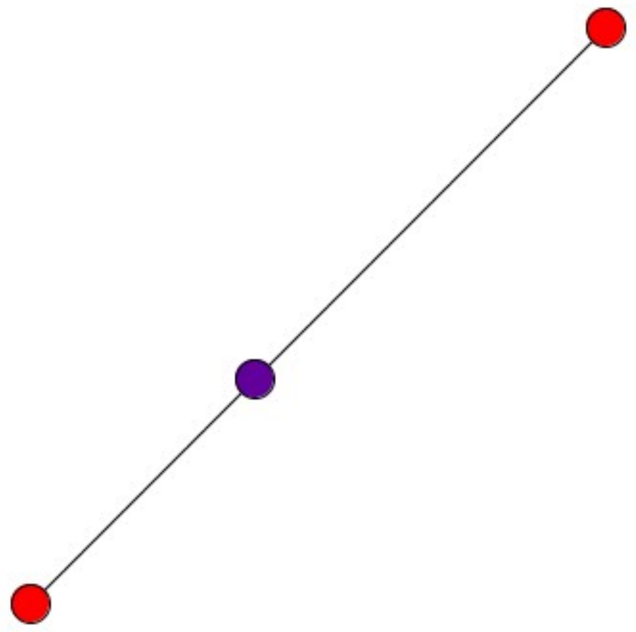
# Review: point-vector line $r = p + tv$

- Line between p (100,400) and q (400,100)
- (y inverted, 0 at top)
- Parametric in t
- Formula in this case?

$$r = (100{,}400) + t * (300, -300)$$

Code:

```
rx = 100 + t * 300;
ry = 400 + t * -300;
```

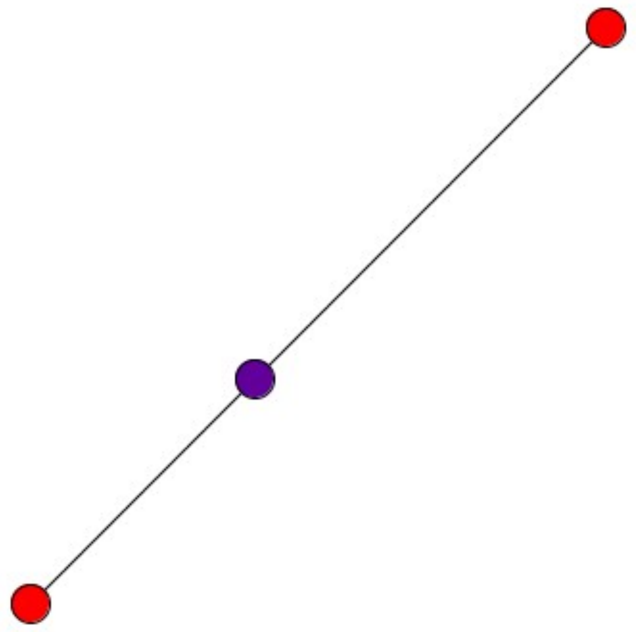


# Review: point-vector line

$$r = p + tv$$

- Processing version

```
void draw () {
  background(255);
  fill(255,0,0);
  line(100,400,400,100);
  ellipse(100,400,20,20);
  ellipse(400,100,20,20);
  float t = map(mouseX,0,width,0,1);
  fill(t*255,0,(1-t)*255);
  float x = 100 + t * 300;
  float y = 400 + t * -300;
  ellipse(x,y,20,20);
}
```



# Review: point-vector line $r = p + tv$

- Unity version

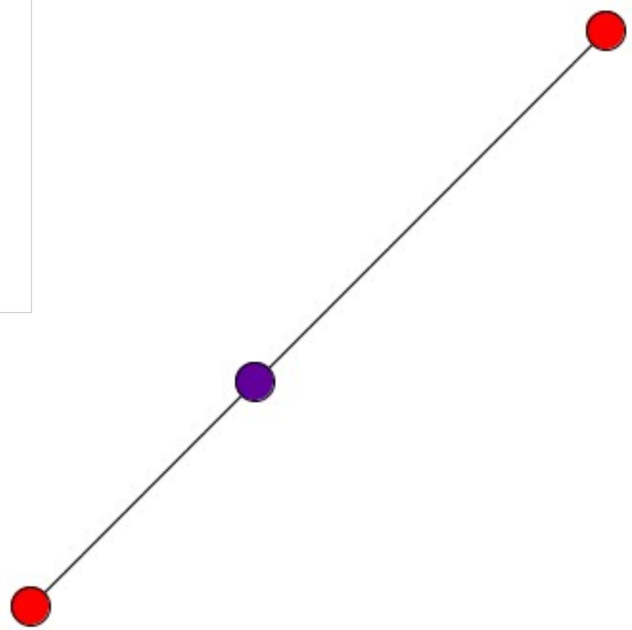
## Vector3.Lerp

public static Vector3 **Lerp**(Vector3 **a**, Vector3 **b**, float **t**);

### Description

Linearly interpolates between two vectors.

```
Vector3 p1 = new Vector(100f,400f,0);
Vector3 p2 = new Vector(100f,400f,0);
Vector3 r  = Vector3.lerp(p1,p2,0.5f);
```



# Lerping to chase

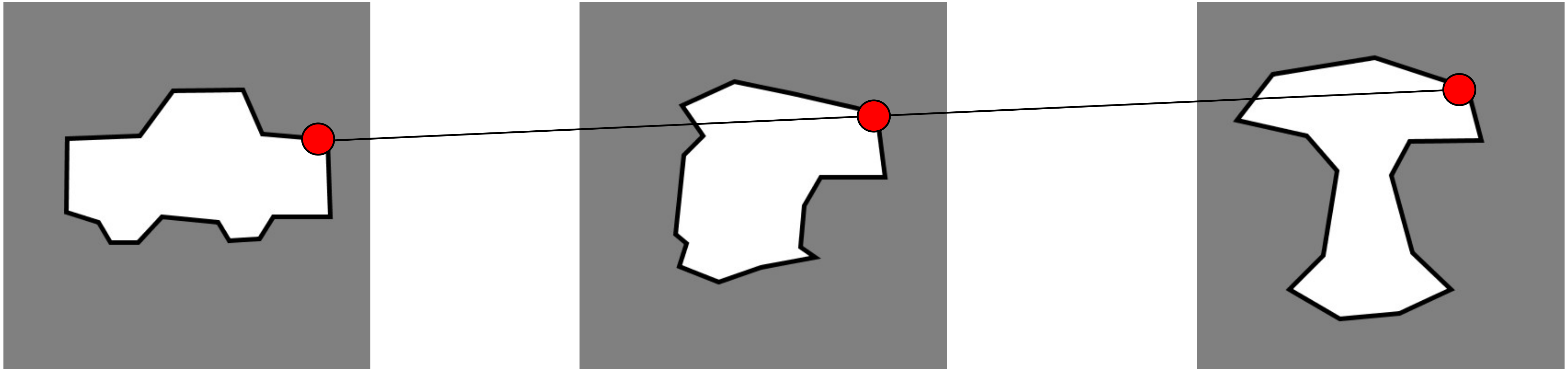
- https://processing.org/examples/interpolate.html

- Go 50% of distance to object chased
- Slows down (*eases*) as you approach

# Lerping to *tween*

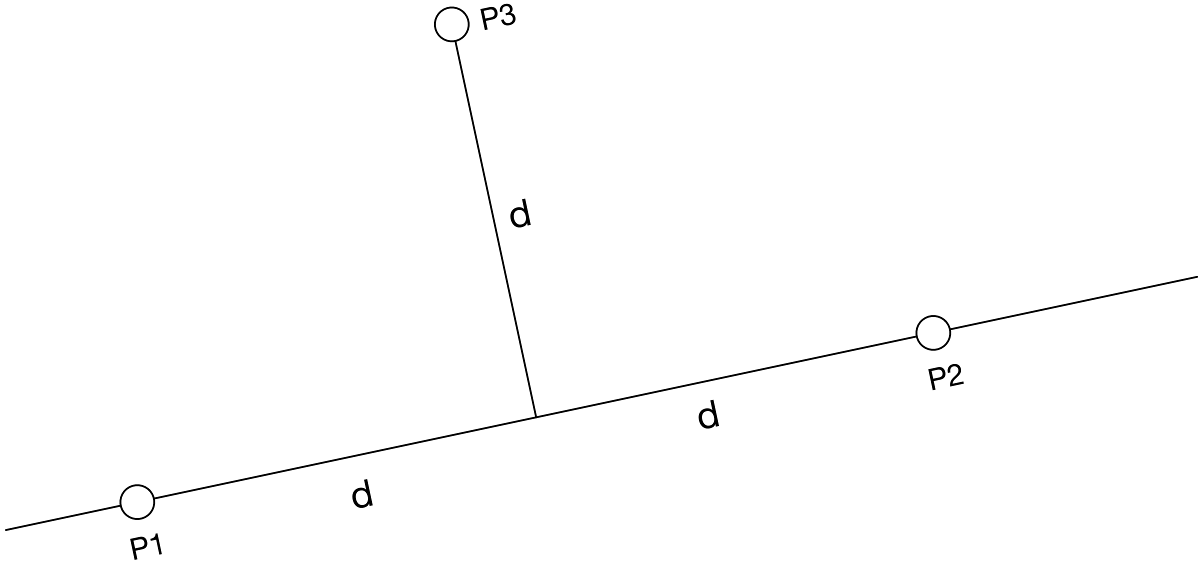
- Interpolate corresponding points on two shapes
- Processing example on website

- Here *polyline*: array of points



# Question 1: Perpendicular bisector?

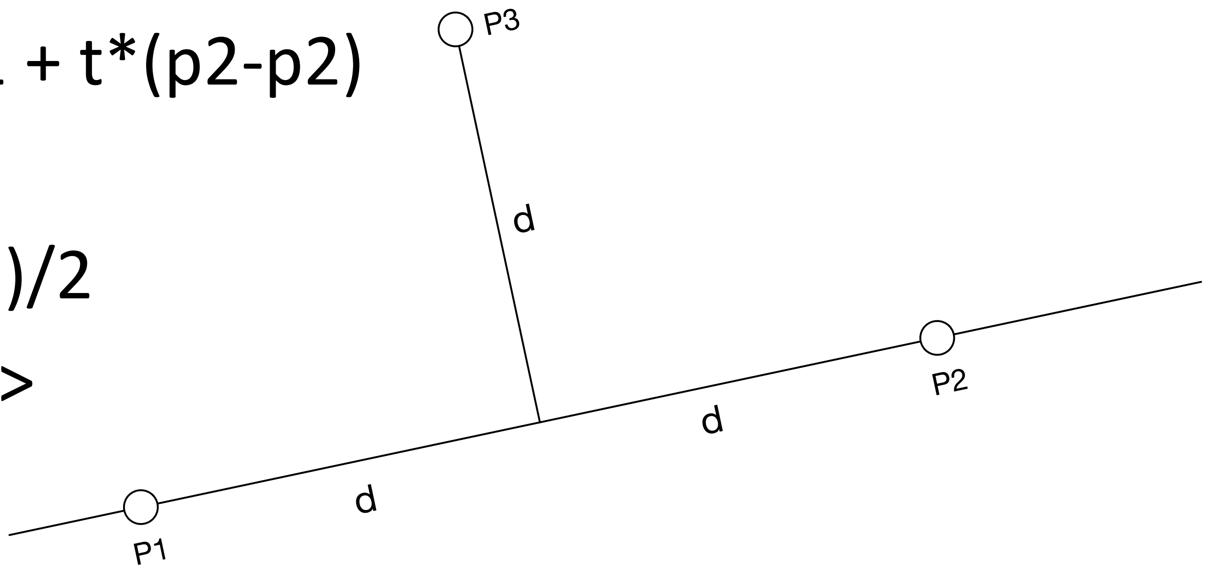
- What's the point-vector form of the line perpendicular to a line segment and through the midpoint? Given p1, p2.



# Question 1: Perpendicular bisector?

- What's the point-vector form of the line perpendicular to a line segment and through the midpoint? Given p1, p2 = (5,10),(30,15)

- Step 1: line p1 to p2 is r(t) = p1 + t*(p2-p2)
- Step 2: Let v = p2-p1
- Step 3: midpoint is m = (p1+p2)/2
- Step 4: perp vector is v' = <-y,x>
- Step 5: r'(t) = m + t * v'



# Question 1: Perpendicular bisector?

- Unity version? Input: p1, p2 Output: p, v in p+tv

- Step 1: line p1 to p2 is r(t) = p1 + t*(p2-p2)
- Step 2: Let v = p2-p1
- Step 3: midpoint is m = (p1+p2)/2
- Step 4: perp vector is v’ = <-y,x>
- Step 5: r’(t) = m + t * v’

## Vector2.Perpendicular

```
public static Vector2 Perpendicular(Vector2 inDirection);
```

# Question 1: Perpendicular bisector?

- Unity version? Input: p1, p2

Output: p, vperp in p+t*verp

- Step 1: line p1 to p2 is r(t) = p1 + t*(p2-p2)
- Step 2: Let v = p2-p1
- Step 3: midpoint is m = (p1+p2)/2
- Step 4: perp vector is v’ = <-y,x>
- Step 5: r’(t) = m + t * v’

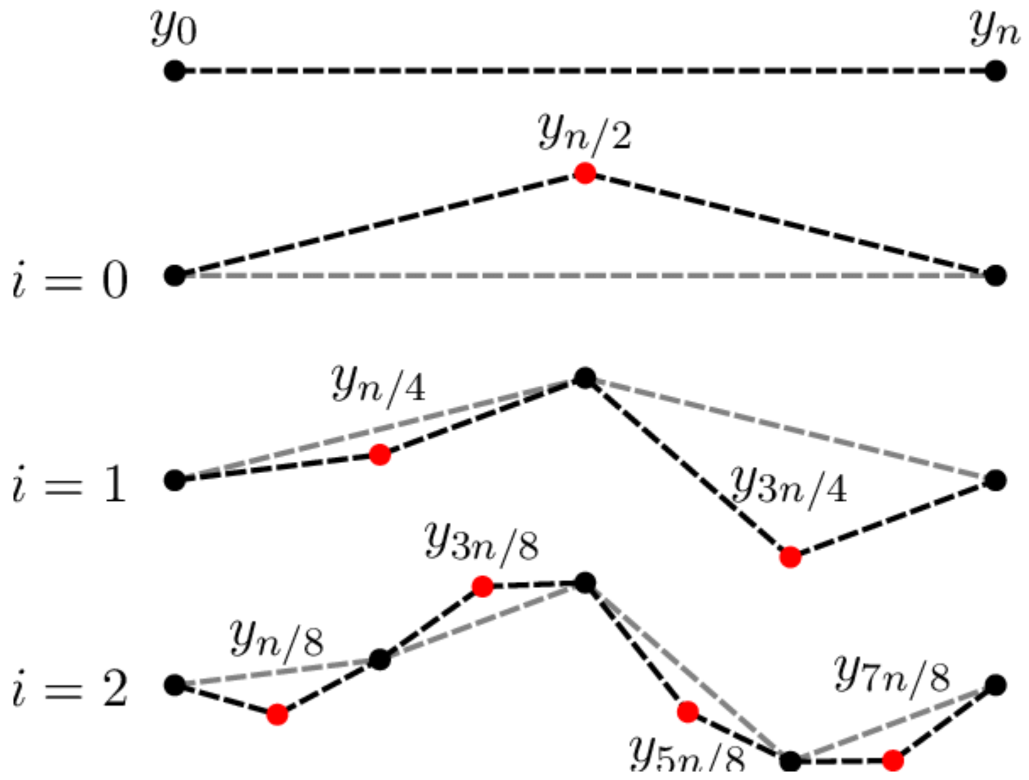
**Vector2.Perpendicular**

public static Vector2 **Perpendicular**(Vector2 **inDirection**);

```
Vector2 m = (p1+p2)/2.0f;
Vector2 v  = p2 – p1;
Vector2 vperp
        = Vector2.perpendicular(v);
// result in m, vperp
```

# Application: midpoint displacement

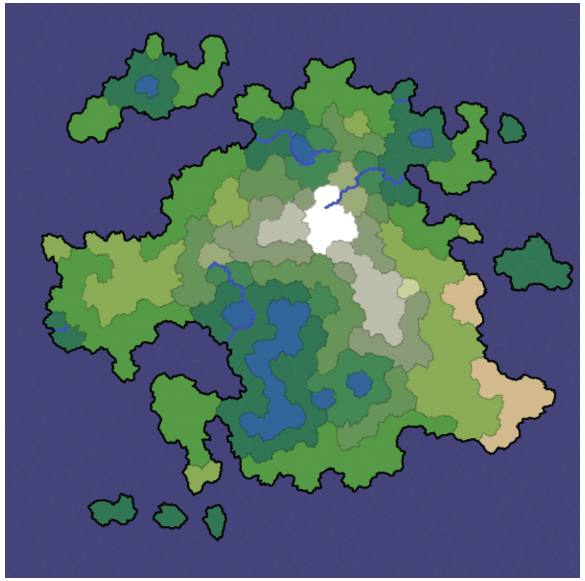
- Recursive curve generation
- Given two points:
  - Create perp bisector
  - Randomly pick t, generate point
  - Repeat for two new line segments
- Works in 3D

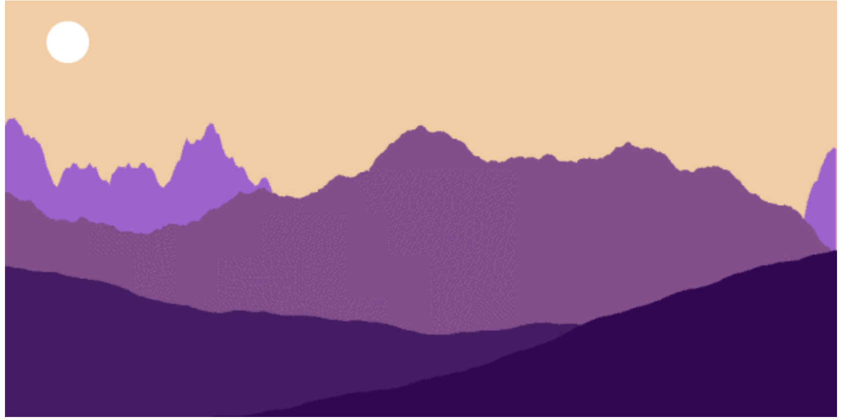


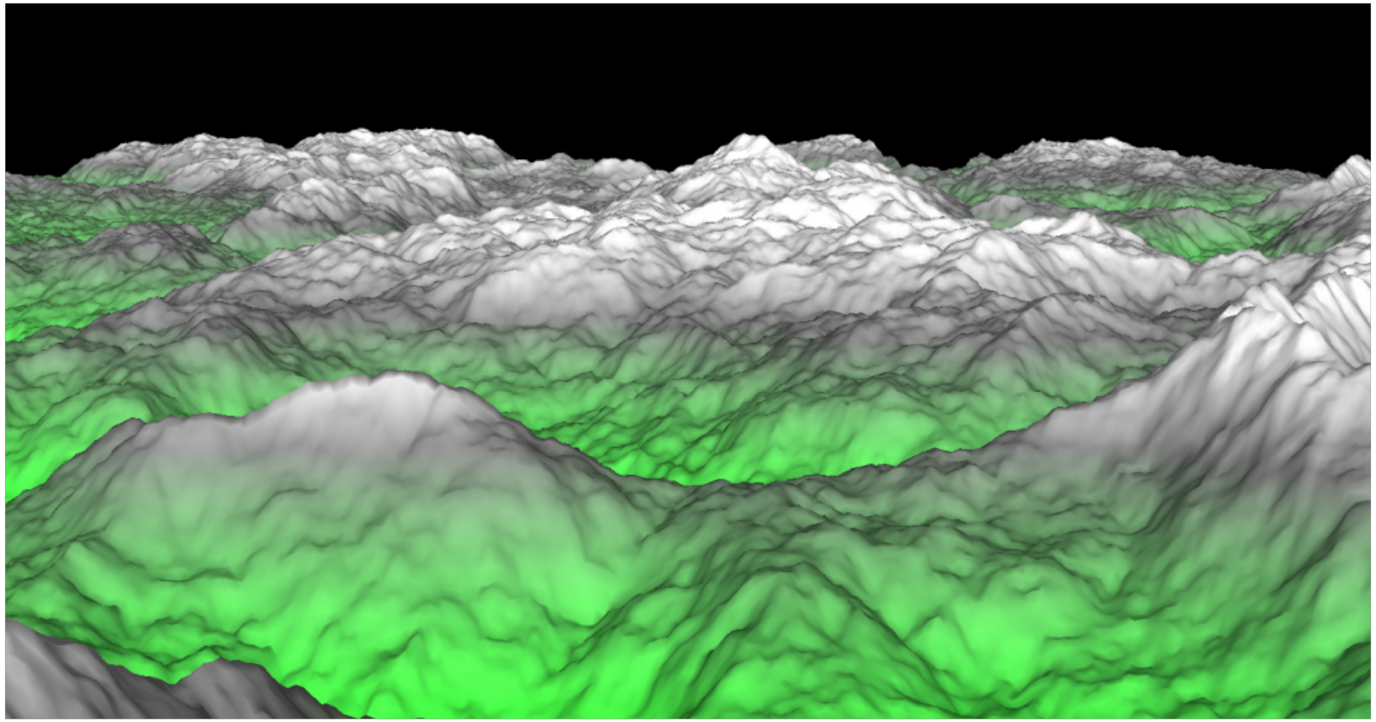

Randomness of t => roughness

# Application: midpoint displacement

- Mountain ranges, terrain, coastlines



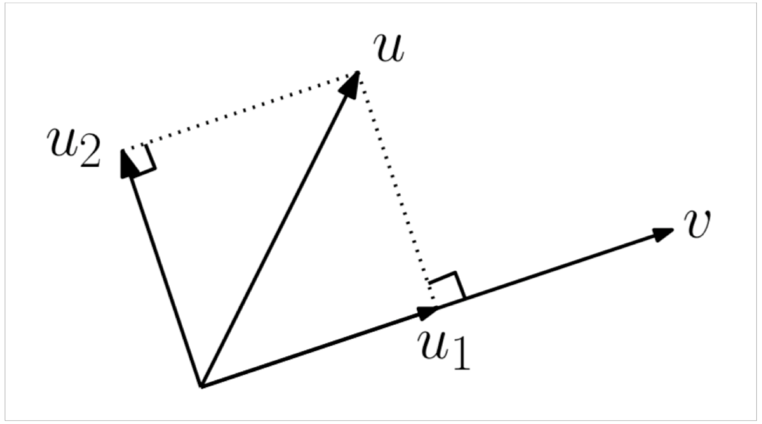




# Back to orthogonal projection

**Orthogonal projection:** Given a vector $\vec{u}$ and a nonzero vector $\vec{v}$, it is often convenient to decompose $\vec{u}$ into the sum of two vectors $\vec{u} = \vec{u}_1 + \vec{u}_2$, such that $\vec{u}_1$ is parallel to $\vec{v}$ and $\vec{u}_2$ is orthogonal to $\vec{v}$.

$$\vec{u}_1 \leftarrow \frac{(\vec{u} \cdot \vec{v})}{(\vec{v} \cdot \vec{v})}\vec{v}, \qquad\qquad \vec{u}_2 \leftarrow \vec{u} - \vec{u}_1.$$



# Problem 10: Find orthogonal projection

- Given p = <1,1> and q=<1,4>, what the orthogonal projection of q onto p?

# Leaving Powerpoint behind …

- To the Chalkboard!

Given vectors $u$, $v$, and $w$, all of type Vector3, the following operators are supported:

```
u = v + w; // vector addition
u = v - w; // vector subtraction
if (u == v || u != w) { ... } // vector comparison
u = v * 2.0f; // scalar multiplication
v = w / 2.0f; // scalar division
```

You can access the components of a Vector3 using as either using axis names, such as, u.x, u.y, and u.z, or through indexing, such as u[0], u[1], and u[2].

The Vector3 class also has the following members and static functions.

```
float x = v.magnitude; // length of v
Vector3 u = v.normalize; // unit vector in v's direction
float a = Vector3.Angle (u, v); // angle (degrees) between u and v
float b = Vector3.Dot (u, v); // dot product between u and v
Vector3 u1 = Vector3.Project (u, v); // orthog proj of u onto v
Vector3 u2 = Vector3.ProjectOnPlane (u, v); // orthogonal complement
```

Some of the Vector3 functions apply when the objects are interpreted as points. Let $p$ and $q$ be points declared to be of type Vector3. The function Vector3.Lerp is short for *linear interpolation*. It is essentially a two-point special case of a convex combination. (The combination parameter is assumed to lie between 0 and 1.)

```
float b = Vector3.Distance (p, q); // distance between p and q
Vector3 midpoint = Vector3.Lerp(p, q, 0.5f); // convex combination
```

# Readings

- David Mount's lecture on Geometry and Geometric Programming