CGT 520 : Libraries and tools

- Building, Debugging
- Window management and basic I/O
- OpenGL extension loading
- Mesh loading
- Image loading
- User interface
- Matrix vector ops

Bottom-up vs. top-down approach

- **Bottom-up**:
  pixel→line→polygon→mesh→scene→lighting…
  - At the end of 16 weeks you can draw a lit cube, woo hoo!!!

- Book utilizes a **top-down** approach
  - Maybe it is more like middle-out…
  - Start with non-trivial working examples, then break them down
  - Make use of libraries that take care of a lot of the busy work
    - Math libs, file loading, user interface and I/O
    - It's good to get used to using/linking/running someone else's code

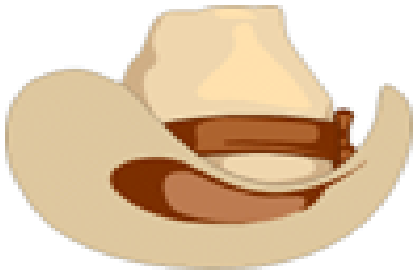
Build and debug environment

- 2017 installed on ECN machines
- Free “Community” versions work fine
- We will use it in lab and for homeworks
- You may choose to work on a different platform, but I can't provide support
- You can demo homework projects to me in class, so bring laptop or arrange remote access to your machine of choice when HW is due

Window management

- http://freeglut.sourceforge.net/
- Original glut lib is abandoned as of 1998
- Window management
- Create OpenGL “rendering context”
- Read keyboard, mouse, joystick using **callback** functions
- Controls display loop
- Portable to Win, Linux, OSX, more...

OpenGL extension loading



- glew: OpenGL extension wrangler
- Supplies code needed to use newer OpenGL functionality on Windows
- Loads new features supplied by some video drivers

Mesh loading

Assimp
Open Asset Import Library

- http://assimp.sourceforge.net/
- No external deps
- loads multiple UVs and vertex colors
- Imports bones, weights, anims
- Many formats: dae, blend, 3ds, ase, obj, ply, md3, lwo, lws...

Image loading

- http://freeimage.sourceforge.net/
- Color conversion, resizing
- Reads and writes many formats: bmp, dds, exr, gif, jpg, png, psd, tga, xpm...
  - Supports more exotic formats, hdr, raw

User interface

- imgui: Immediate mode graphical user interface
- https://github.com/ocornut/imgui
  - Sliders
  - Buttons
  - Text entry
  - Color picker
  - Good for debugging

Matrix/Vector Operations

- http://glm.g-truc.net/
- Matrix, vector ops
- Functions to help pass variables into shaders

Matrix/Vector Operations

- Scalars, vectors, matrices
- Operations, graphical interpretation, glm code
  - Vector-vector addition
  - Scalar-vector multiplication
  - Dot product
  - Cross product
  - Matrix-vector multiplication
  - Matrix-matrix multiplication
  - Matrix inverse

Scalars, vectors, matrices

- **Scalars** : a single number representing magnitude
  - Length, distance, speed
  - Mathematical notation: α, β, γ
  - In code : `float a,b,c;`

Scalars, vectors, matrices

- **Vector** : a quantity with both magnitude and direction
  - Displacement, velocity
  - Mathematical notation: $u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, \quad v = \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix}$
  - In code: `glm::vec3 u, v;`

Scalars, vectors, matrices

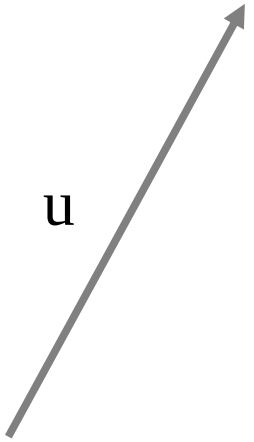
- **Matrix** : a quantity representing a linear transformation
  - Rotation, translations
  - Mathematical notation:

$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

  - In code: `glm::mat3 M;`

Vector magnitude

$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}$$



$$||u|| = \sqrt{u_x^2 + u_y^2 + u_z^2}$$

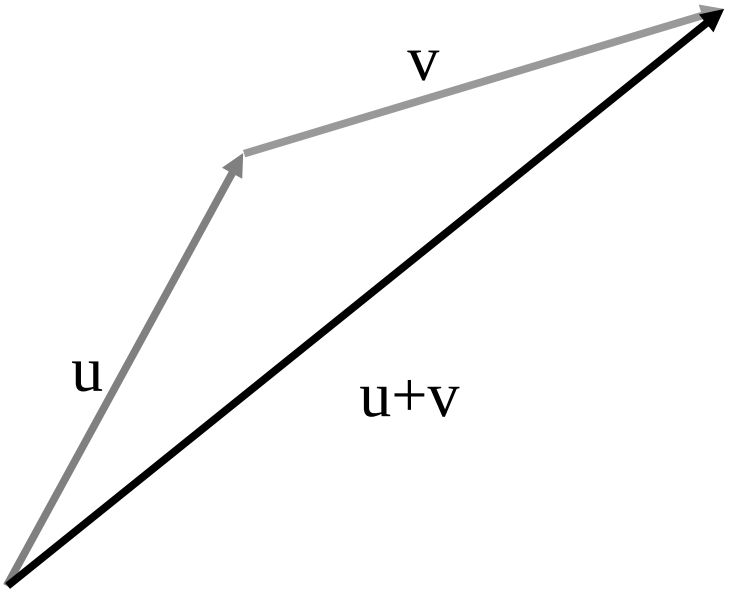
```
float len = glm::length(u);
```

Vector-vector addition

- "Head-to-tail" rule

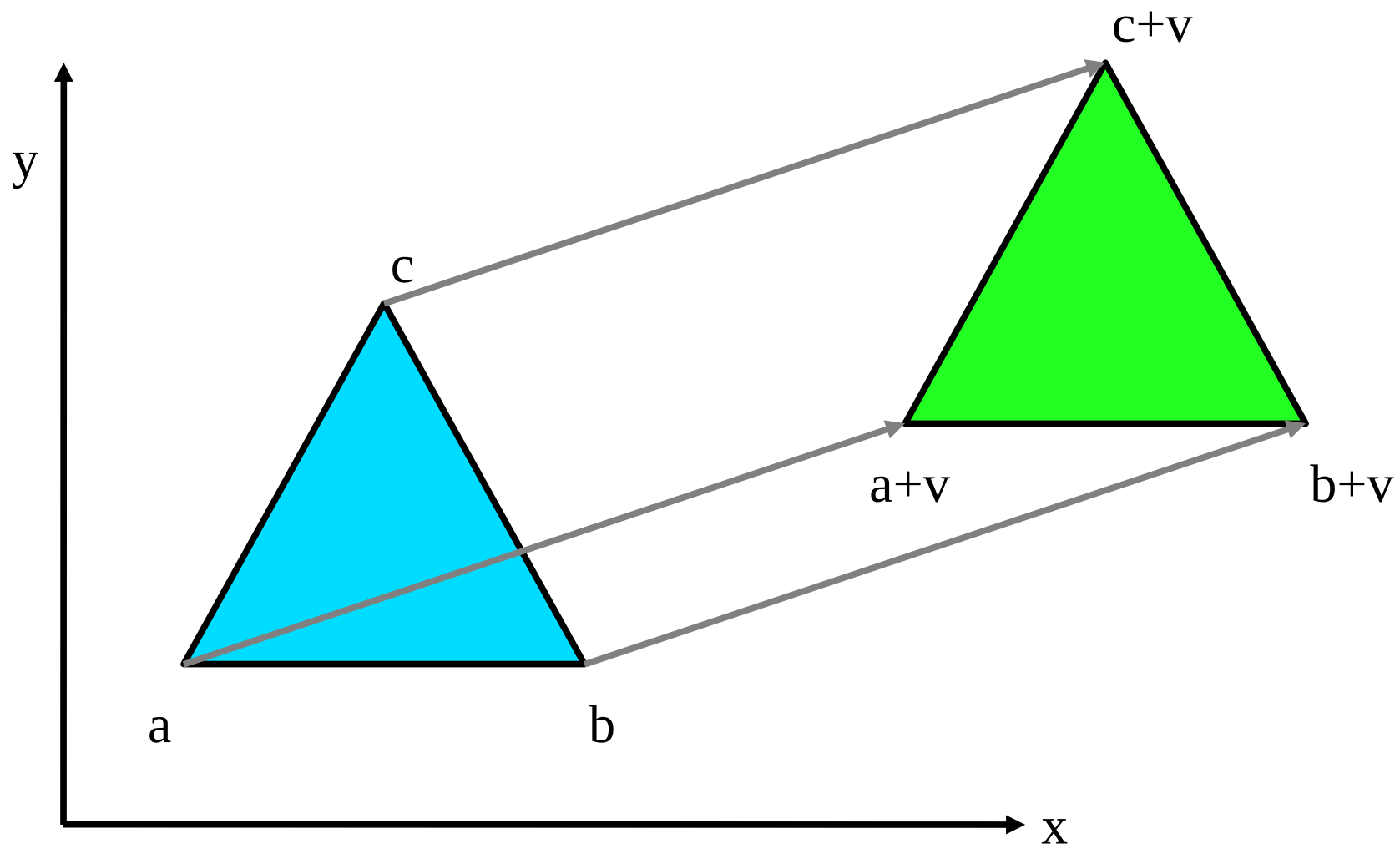
$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

$$u + v = \begin{bmatrix} u_x + v_x \\ u_y + v_y \\ u_z + v_z \end{bmatrix}$$



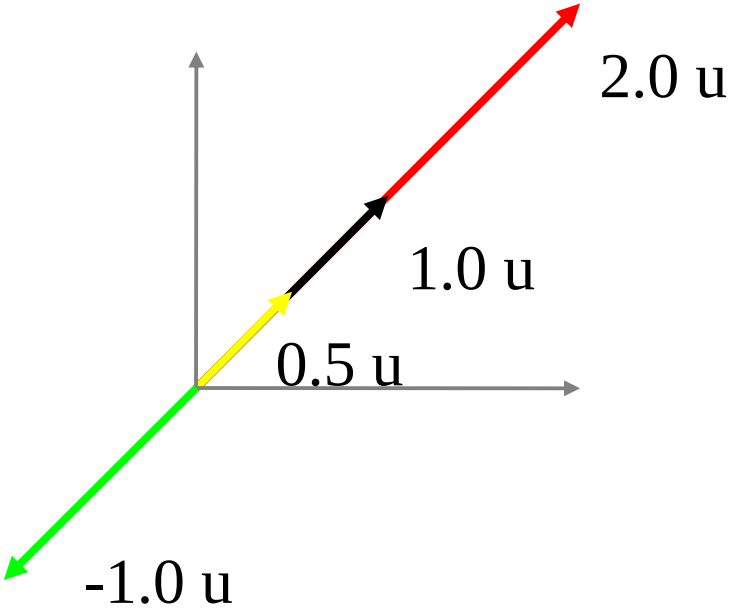

```
glm::vec3 w = u+v;
```

Point-vector addition



Scalar-vector multiplication

$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, \alpha u = \begin{bmatrix} \alpha u_x \\ \alpha u_y \\ \alpha u_z \end{bmatrix}$$

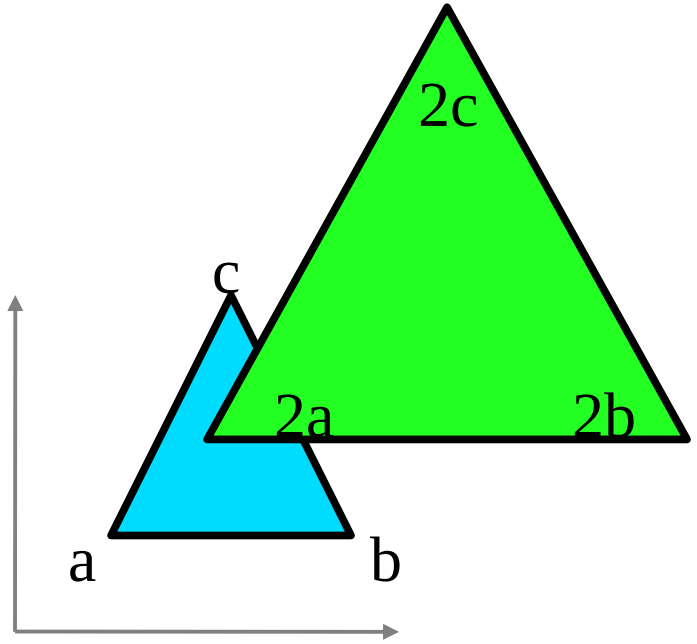


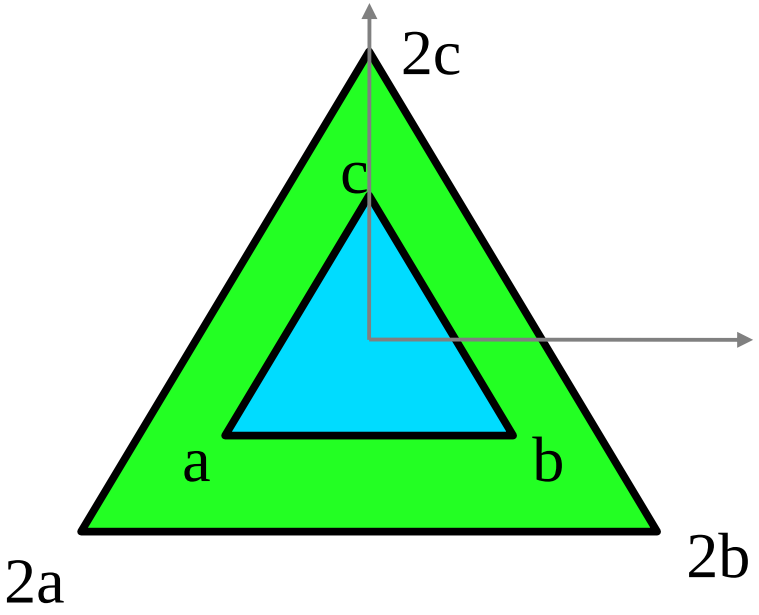
- For α > 1, length increases
- For 0 < α < 1, length decreases
- For α < 0, vector is reflected and scaled

```
glm::vec3 w = a*v;
```

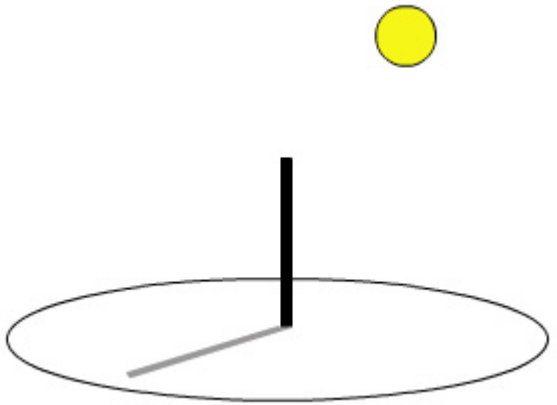
Scalar-vector multiplication

- Uniform scaling about the origin





Projection

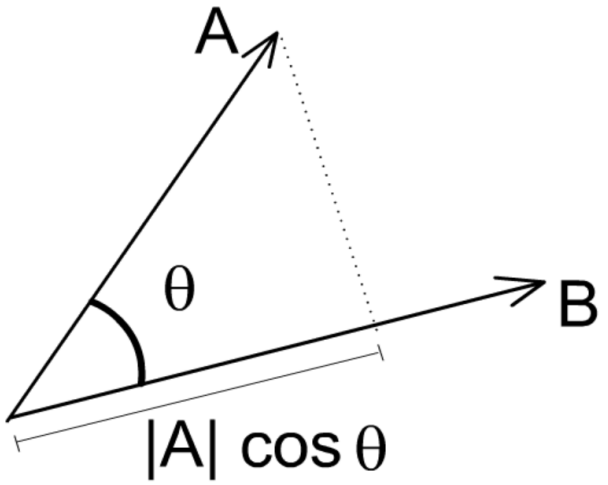


Roughly definition: Taking an object and reducing its dimensionality.

The shadow of a 3D object is a 2D projection of its shape.

Projections

Projecting 2D vector A onto B:



Cosine function and the related **dot product** operator show up in many contexts:

- Lighting
- Collision detection
- Coordinate transformations

Dot product

- Sometimes called “inner product”
- The result is a scalar

$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix} \qquad u \cdot v = u_x v_x + u_y v_y + u_z v_z$$

- *Can use to compute vector magnitude*

$$||u|| = \sqrt{u \cdot u}$$

- *Can use to compute angles*

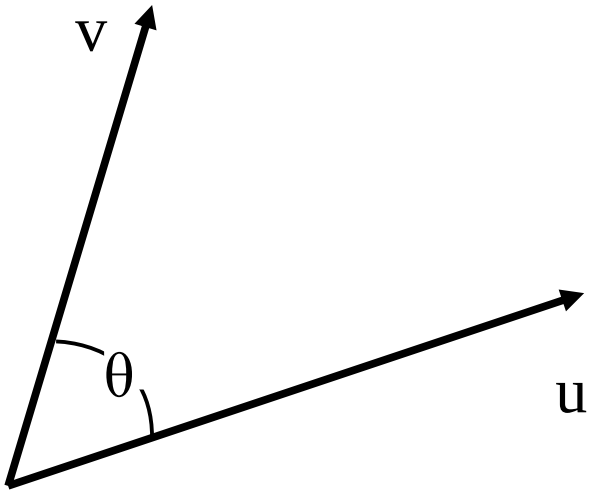
$$u \cdot v = ||u||||v|| \cos\theta$$

```
float w = glm::dot(u,v);
```

Dot product

- Computing angles with the dot product

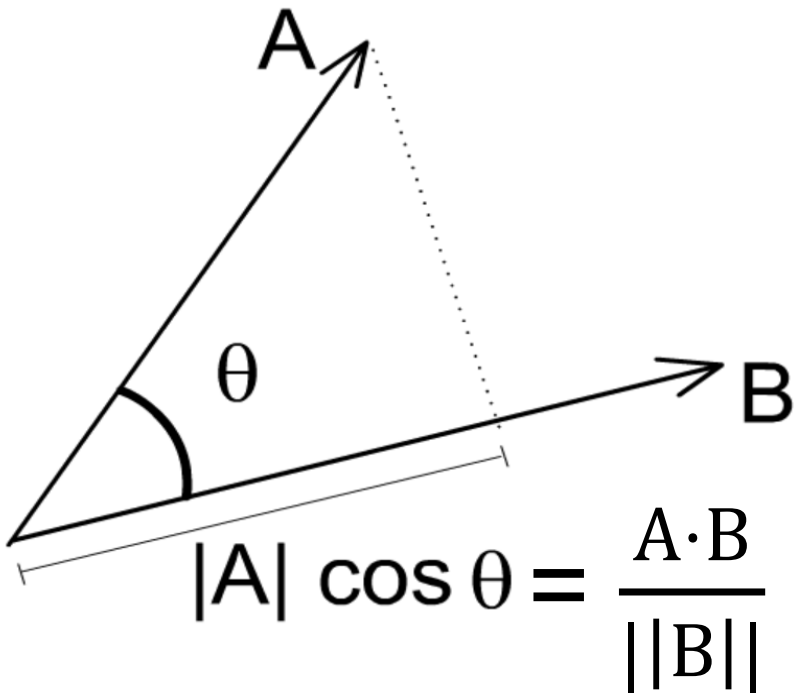
$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$



$$u \cdot v = \left||u|\right|\left||v|\right| \cos\theta$$

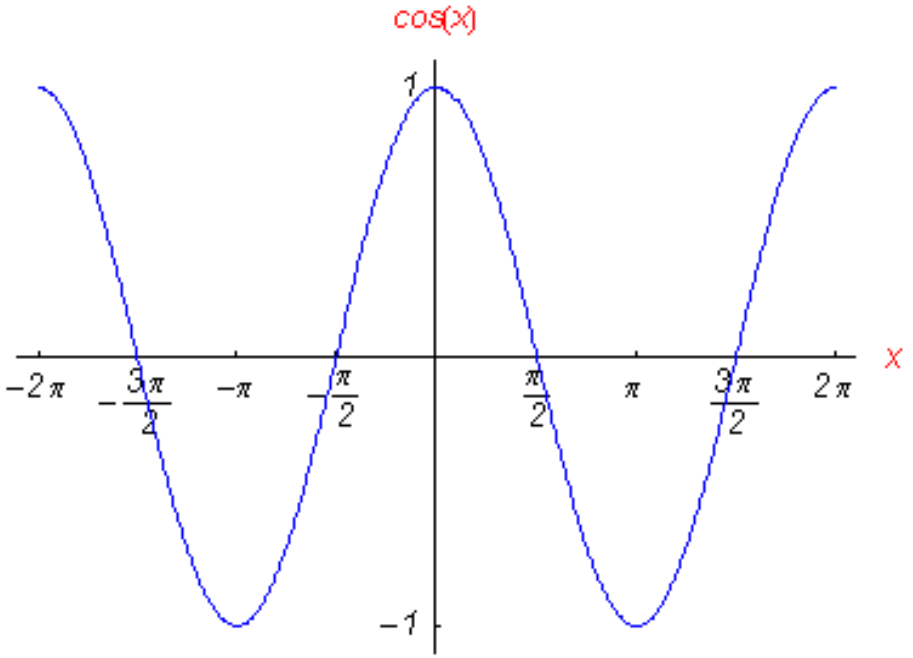
$$\cos\theta = \frac{u \cdot v}{\left||u|\right|\left||v|\right|}$$

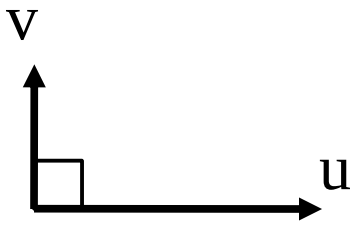
Projections



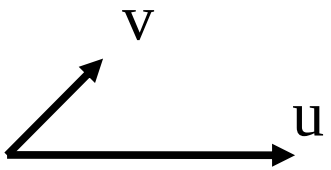
Dot product : Projections

- Sign of dot product depends on relative direction





$u \cdot \mathrm{v} = 0$

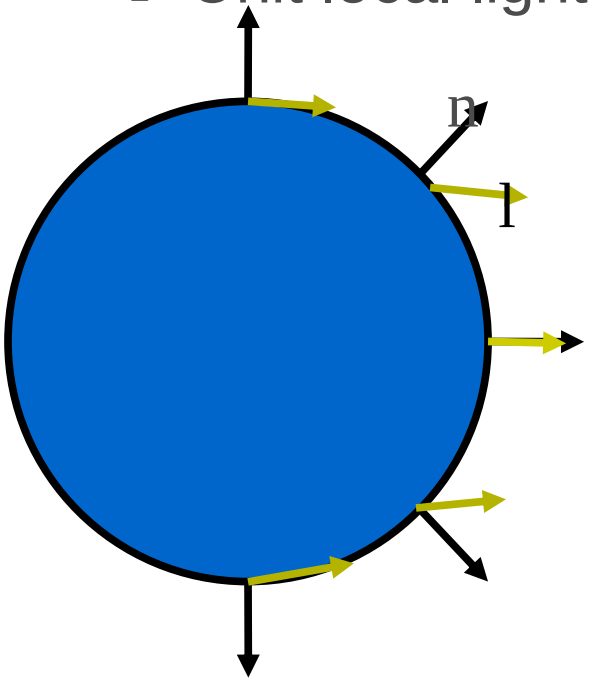


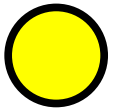
$u \cdot \mathrm{v} > 0$

$u \cdot \mathrm{v} < 0$

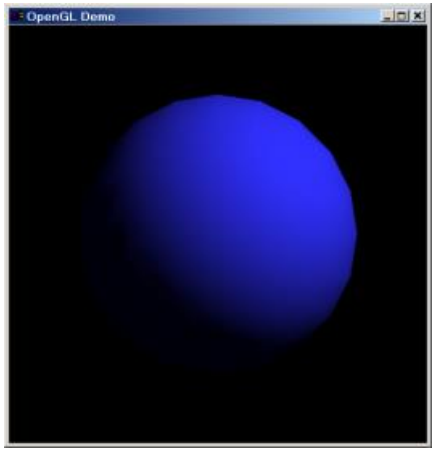
Dot product : Lighting

- At each point on the surface:
  - Unit normal, n is perpendicular to surface
  - Unit local light vector, l, points toward light source





light

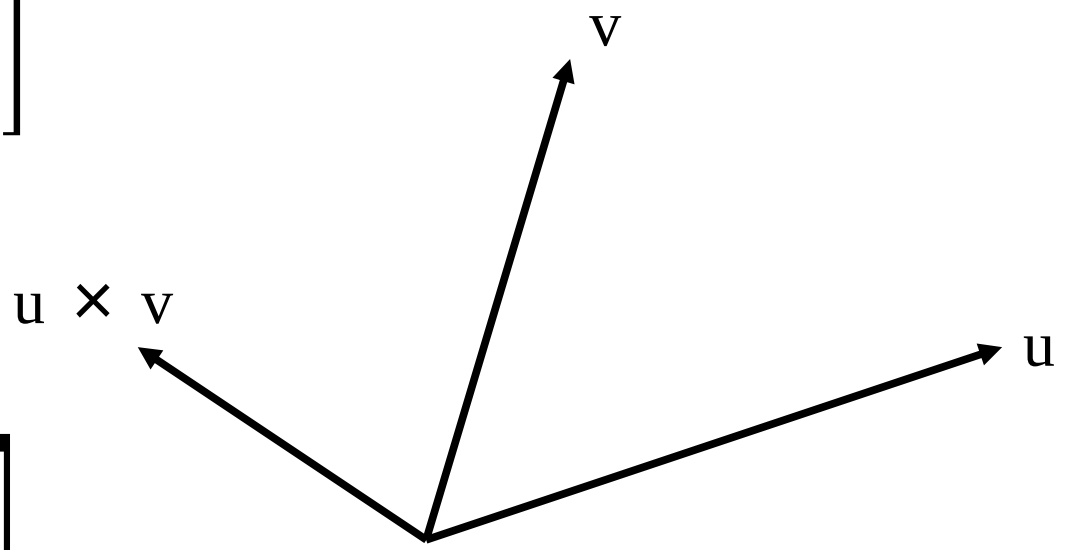


$$\text{Intensity} = \max(0.0, n \cdot l)$$

Cross product

- The result is a vector **perpendicular** to the plane u,v lie in

$$u = \begin{bmatrix} u_x \\ u_y \\ u_z \end{bmatrix}, v = \begin{bmatrix} v_x \\ v_y \\ v_z \end{bmatrix}$$

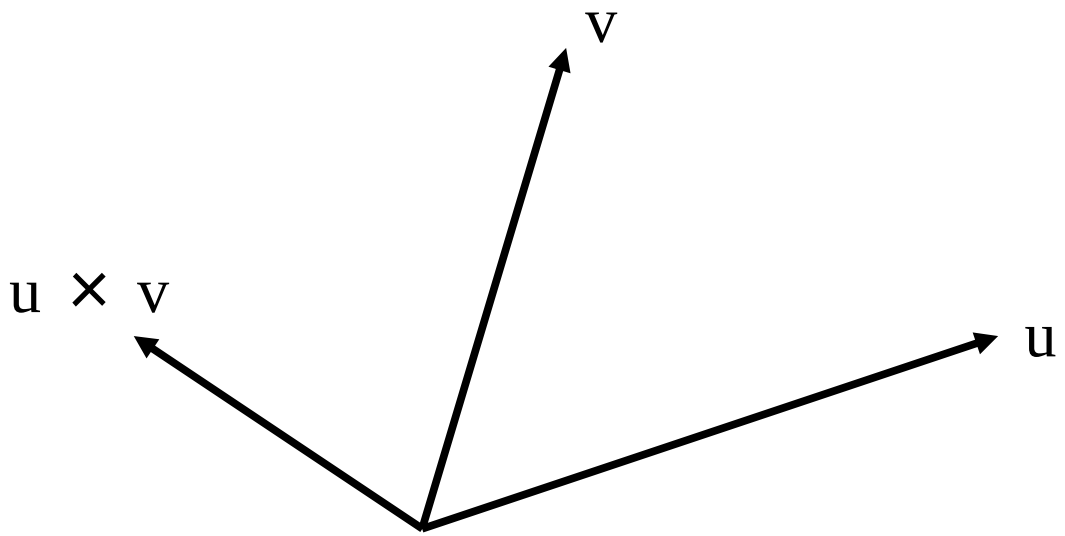


$$u \times v = \begin{bmatrix} u_y v_z - u_z v_y \\ u_z v_x - u_x v_z \\ u_x v_y - u_y v_x \end{bmatrix}$$

Recall – “right-hand rule”

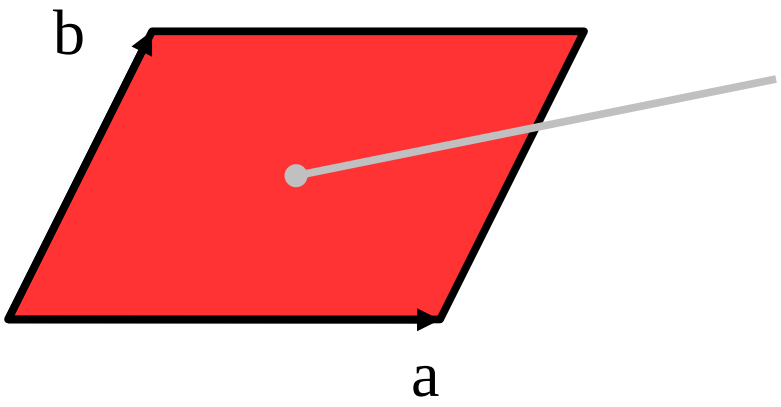
```
glm::vec3 w = glm::cross(u, v);
```

- What is $u \times v$ if $u = v$?
- What is $u \cdot (u \times v)$?

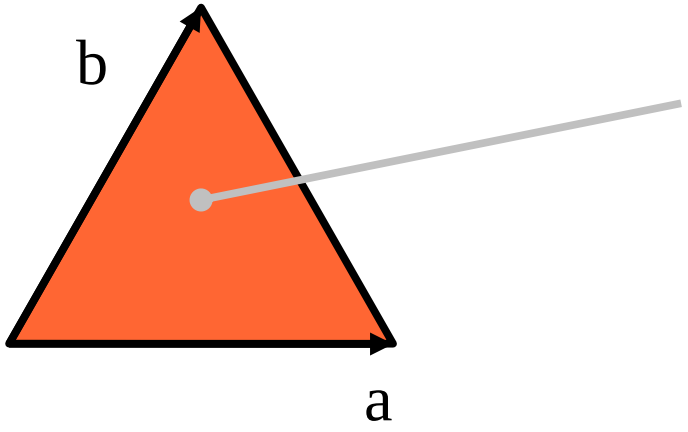


Cross product

- Area of parallelogram (twice the area of the triangle)



$$\text{Area} = \|a \times b\|$$



$$\text{Area} = \tfrac{1}{2} \|a \times b\|$$

Matrix-vector multiplication

- The result is a vector

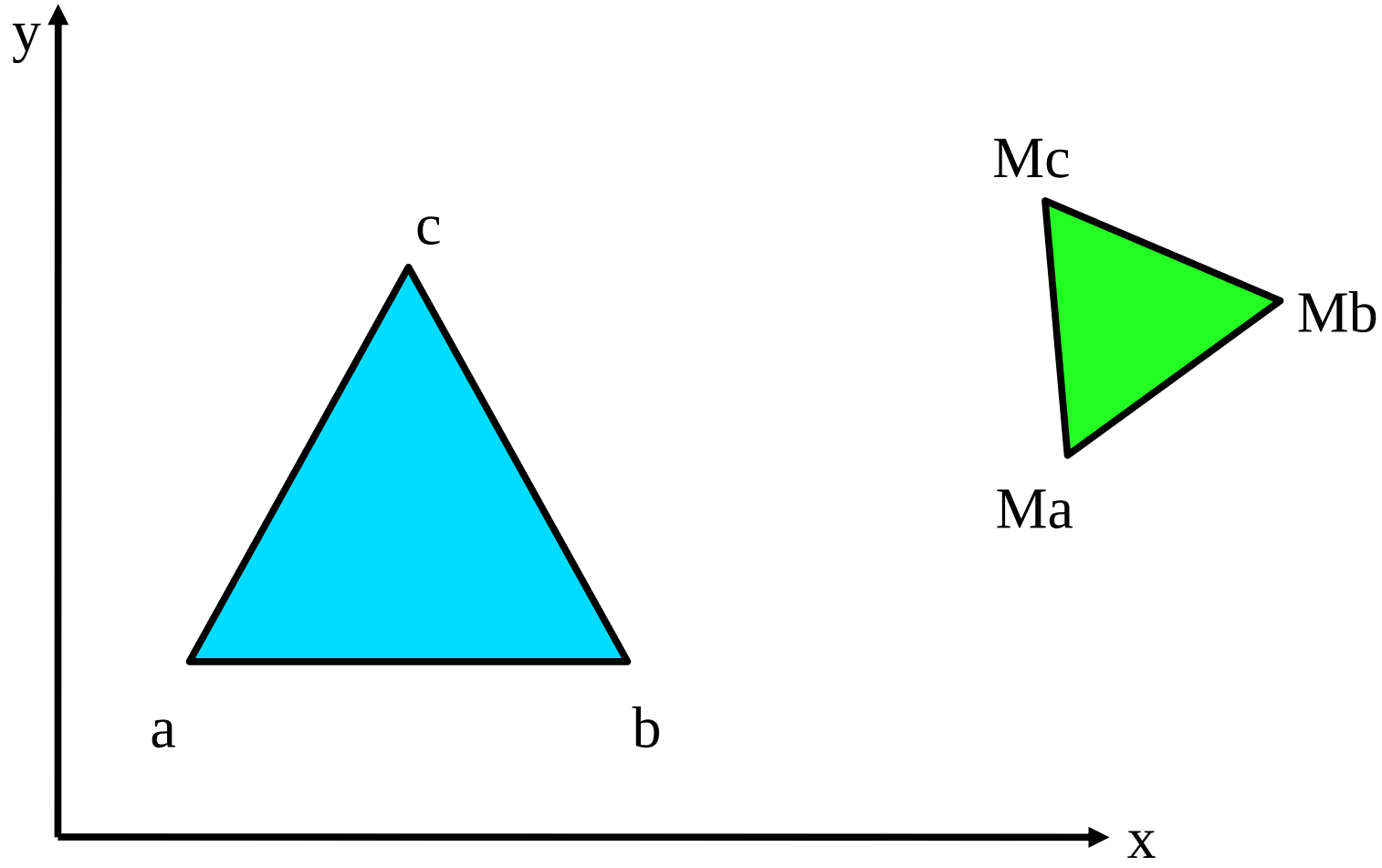
$$M = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix}$$

$$Mv = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} M_{11}v_1 + M_{12}v_2 + M_{13}v_3 \\ M_{21}v_1 + M_{22}v_2 + M_{23}v_3 \\ M_{31}v_1 + M_{32}v_2 + M_{33}v_3 \end{bmatrix}$$

```
glm::vec3 u = M*v;
```

Linear transformations

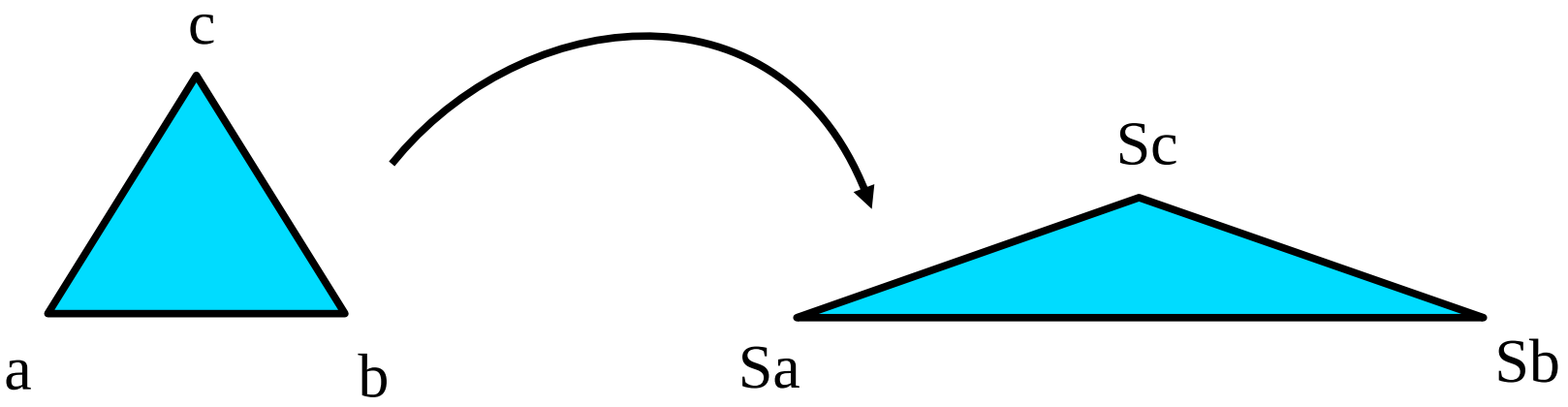
- Rotation, translation, scale and more...



Nonuniform scaling

$$S = \begin{bmatrix} S_1 & 0 & 0 \\ 0 & S_2 & 0 \\ 0 & 0 & S_3 \end{bmatrix}$$

$$Sv = \begin{bmatrix} S_1 & 0 & 0 \\ 0 & S_2 & 0 \\ 0 & 0 & S_3 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \\ v_3 \end{bmatrix} = \begin{bmatrix} S_1 v_1 + 0 + 0 \\ 0 + S_2 v_2 + 0 \\ 0 + 0 + S_3 v_3 \end{bmatrix} = \begin{bmatrix} S_1 v_1 \\ S_2 v_2 \\ S_3 v_3 \end{bmatrix}$$



Matrix ops in glm

- Multiplication

```
glm::mat3 C = A*B;
```

- Inverse

```
glm::mat3 D = glm::inverse(C);
```

- Transpose

```
glm::mat3 E = glm::transpose(D);
```

Wrap up

☺ Top-down approach and code libraries will allow us to start working with a nontrivial graphics application early on

☹ Have to learn a bit about each of these libraries