
GDC
Rigging with Triangles
Visualizing the Vectors and Shapes in Rigging
Richard Katz
Senior Technical Artist
Blizzard Entertainment
GAME DEVELOPERS CONFERENCE® | MARCH 19-23, 2018 | EXPO: MARCH 21-23, 2018 #GDC18
UBM

Hello!

Richard Katz

- Senior Technical Artist, Blizzard Entertainment
- World of Warcraft

- 20 years in Game Development
- 7 years as an Artist, Animator, Lead
- 12+ years as Technical Artist
- Second time at TABC

SIERRA
3DO
BLIZZARD ENTERTAINMENT
Zenimax online
VISUAL CONCEPTS

I'm Richard Katz, Senior Technical Artist on the World of Warcraft team at Blizzard Entertainment in Irvine, CA. I've been at Blizzard for 3 years, but I've been a professional Game Developer for over 20 years. I spent my first 7 years as a 3D Character Artist, Animator, and Lead Artist. I've spent the past 12+ years as Technical Artist specializing in Character Rigging and Animation Tools.

Thanks for coming. This is my second time speaking at the Tech Art Boot Camp, the first time was in 2015

Rigging with Triangles

What is this talk about?

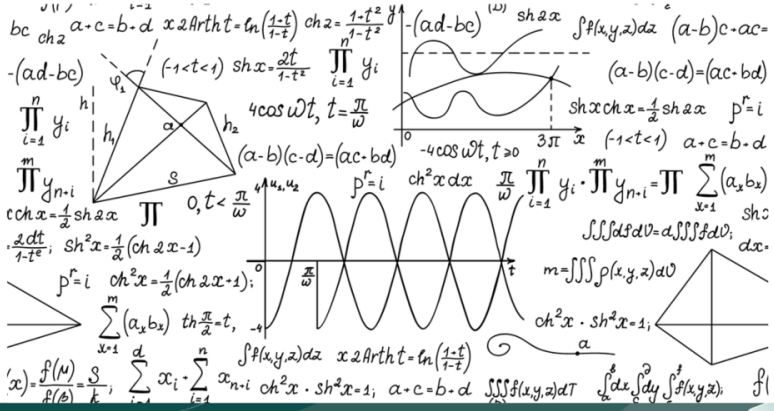
Not too long ago, I had heard the terms “dot product”, “cross product”, “transformation matrix” but I didn’t really know what they were or how to use them. Only about 6 or 7 years ago, something finally clicked and I figured out just enough to get myself into trouble. But I thought if I’m just figuring this out now, everyone else must already have mastered this stuff, right?

After putting together a python matrix module at work, I ran a quick demo and tutorial for my team, and realized that even some really smart people I

work with never really had to deal with matrices and vectors that much. They appreciated the way I broke it down for them and told me they understood it a little bit better after the meeting. This talk evolved out of what I put together for that demo.

Rigging with Triangles

Refresher on Linear Algebra



There will be some math, but don't panic. What I want to do is take those ugly lists of numbers, and try to show you that you can visualize them as pretty arrows and triangles, and that they can mean the same thing. If I can figure this stuff out, then you can too!

Bend your index finger. There are equations that can describe that motion, and how the parts relate to each other. It all starts with math.

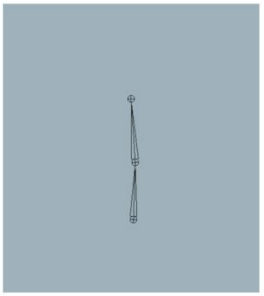
Rigging with Triangles

Visualize it in a new way

For some of you, this might be old news, but I hope I can try to help some people visualize it in a new way

Rigging with Triangles

Application in Rigging



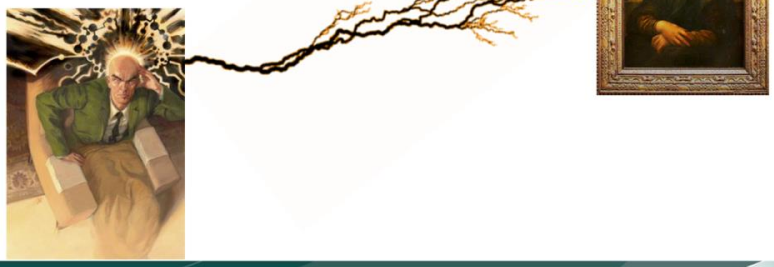
1. Make some joints.

2. Rig the rest of the Owl.

I’m going to take those basic concepts and put them together into a couple of example rigging applications.

Rigging with Triangles

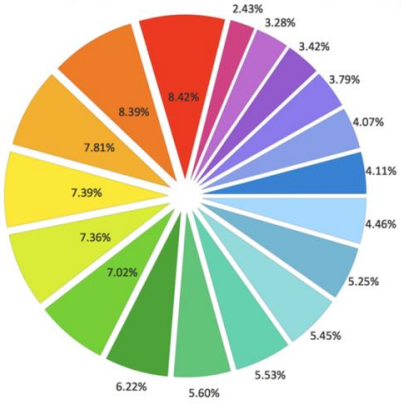
Art



I come from an art background. Some rare artists can put a pen to paper and create a fully-formed image from their mind through their hand and onto the page

Rigging with Triangles

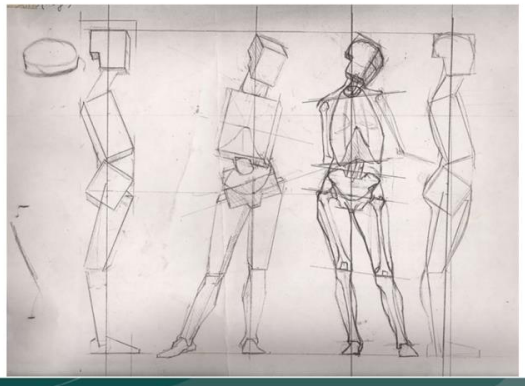
Break a problem down into manageable parts



But complex tasks usually require one to break a problem down into manageable parts. Character Rigging is a series of Problems to be solved. Like any problem, first we analyze it. Then we break it down to smallest practical parts, and solve those smaller bits. Finally we put the parts back together.

Rigging with Triangles

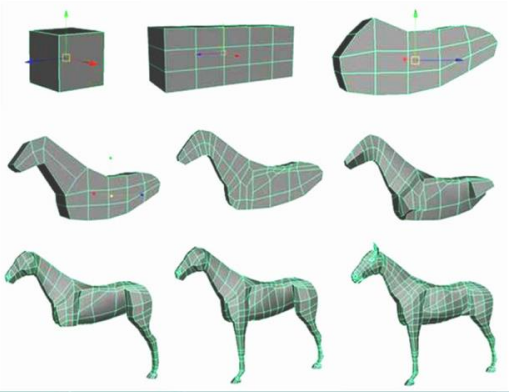
Figure drawing: basic shapes



When teaching figure drawing, an artist learns how to break a character into basic primitive shapes: Cylinders, Spheres, and Cubes.

Rigging with Triangles

3D: Build from primitives

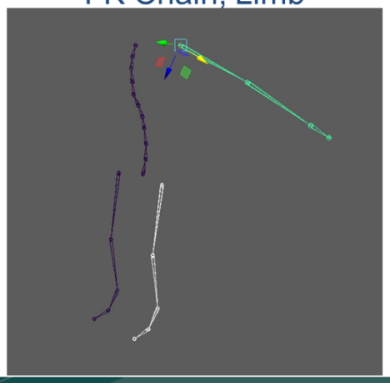


When I was building 3D Models, I worked in very much the same way, with actual primitive shapes that I molded and welded together.

Rigging with Triangles

Rigging building blocks:

FK Chain, Limb



In rigging, I'm similarly putting together building blocks: such as an FK Chain or a Limb.

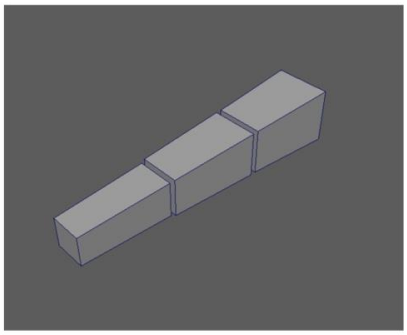
Rigging with Triangles

"Build Finger"

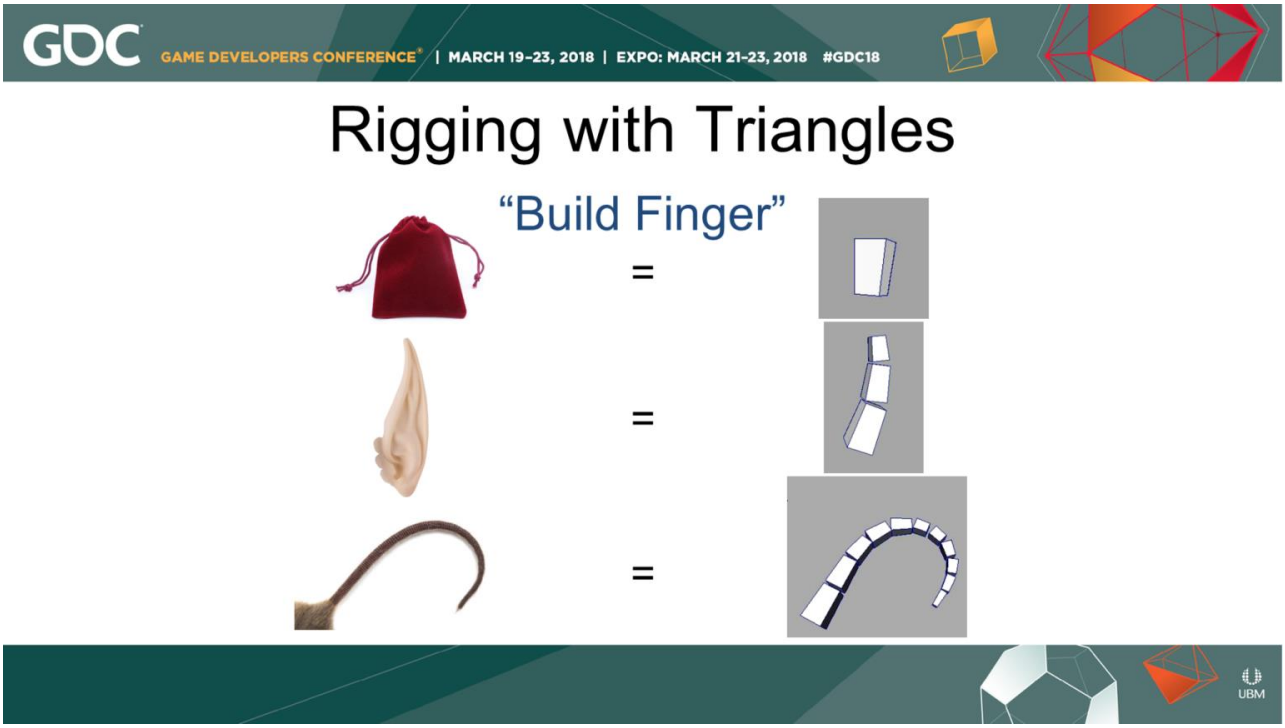
Let me tell you a story. When building rigs for my last project, I started with a root control, a hip control, a spine, a head, arms, legs.

Rigging with Triangles

"Build Finger"



Then I got to the fingers, and I wanted to make a single function that built controls for all 5 fingers. 1 joint fingers, 2 joint fingers, 3 joint fingers.



Eventually, I wanted to make little extra bits on characters: A pouch on their belt, a floppy bit on their elbow, even some secondary motion on elf ears. I had everything I needed in my "finger" component, so I ended up using it for every little FK bit on many characters.

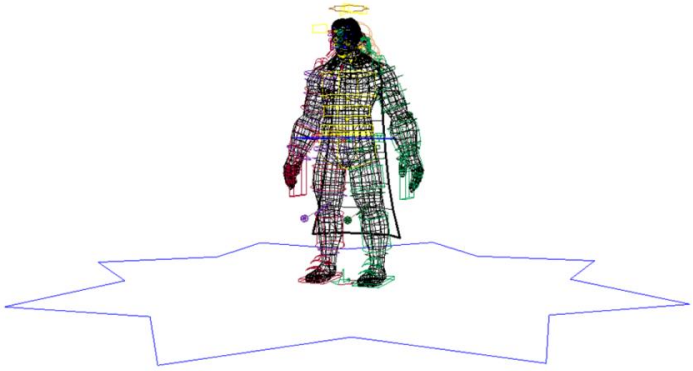
Rigging with Triangles

"Build Finger"

The "Finger" ended up being the building block of my rig.

Rigging with Triangles

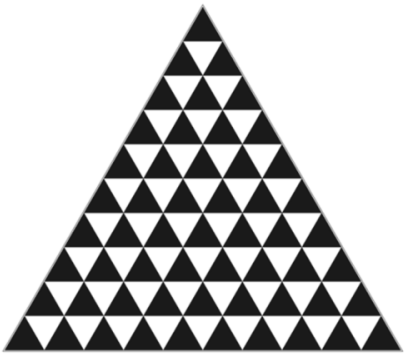
```
global proc buildFingerControl(...
```



When I came to my current team, I was somewhat relieved to see that they had done the same thing! Every FK control on our “modern” rigs is build by a function called “buildFingerControl”. Even FK spines, tails, head and neck, all are built with that function.

Rigging with Triangles

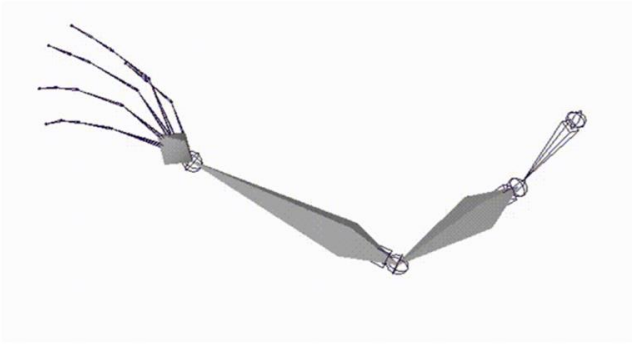
Triangle



Even more basic than a fully-functioning rig module is how I came to break it down into an even more basic element: The Triangle.

Rigging with Triangles

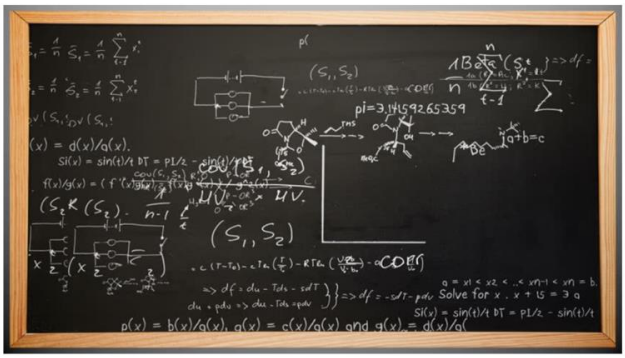
Triangle



When I look at potential rig components, I tend to visualize them as triangles. This allows me to apply some math to the situation and produce an accurate solution. Before we get ahead of ourselves, let's go over some of the math to get from here to there.

Rigging with Triangles

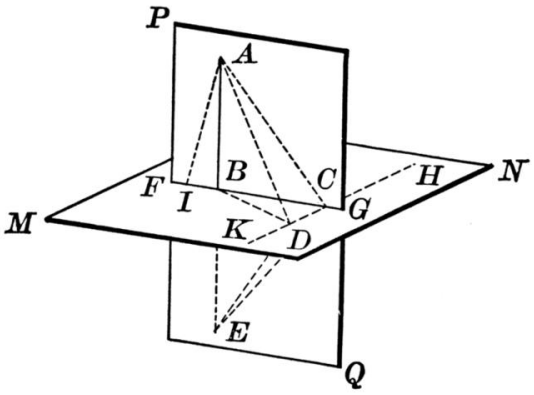
Math



Let’s talk about Math. Since I went to school for art, I didn’t have to take many math courses, and I probably would have forgotten most of it anyway. So I had to learn most of this by piecing it together over time.

Rigging with Triangles

Linear Algebra



I'll be talking about math in how it relates to character rigging, but much of the math is also directly applicable to other sub-disciplines of Technical Art: writing shaders, for instance. What is Linear Algebra?

Rigging with Triangles

Linear Algebra

Wikipedia says: Linear algebra is the branch of mathematics concerning vector spaces and linear mappings between such spaces. It includes the study of lines, planes, and subspaces, but is also concerned with properties common to all vector spaces.

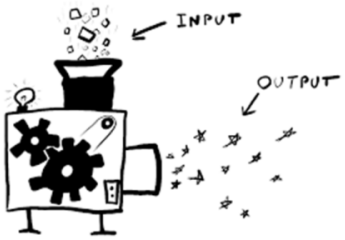
Rigging with Triangles

Linear Algebra

To paraphrase, it is the manipulation of various mathematical structures that retains the functions of addition and multiplication

Rigging with Triangles

Linear Algebra



I tend to think of of these “tools” I use on a daily basis as black boxes: transform matrix, cross and dot products. I put values in, I get results out.

Rigging with Triangles

Linear Algebra

$$
\begin{aligned}
\mathbf{c} \cdot \mathbf{c} &= (\mathbf{a} - \mathbf{b}) \cdot (\mathbf{a} - \mathbf{b}) \\
&= \mathbf{a} \cdot \mathbf{a} - \mathbf{a} \cdot \mathbf{b} - \mathbf{b} \cdot \mathbf{a} + \mathbf{b} \cdot \mathbf{b} \\
&= a^2 - \mathbf{a} \cdot \mathbf{b} - \mathbf{a} \cdot \mathbf{b} + b^2 \\
&= a^2 - 2\mathbf{a} \cdot \mathbf{b} + b^2 \\
c^2 &= a^2 + b^2 - 2ab\cos\theta
\end{aligned}
$$

You can use them even if you don't memorize underlying math as long as you know how to use them.

Rigging with Triangles

Linear Algebra

Just like artists who use scripts don't have to know how to write scripts on their shelf to know which one to click to produce the desired result.

Rigging with Triangles

Vectors and Matrices

What is a Vector, Victor?

Some of the mathematical tools I use when solving those rigging problems are Vectors and Matrices. What IS a vector?

Rigging with Triangles

Vectors

A direction

It is a Direction

Rigging with Triangles

Vectors

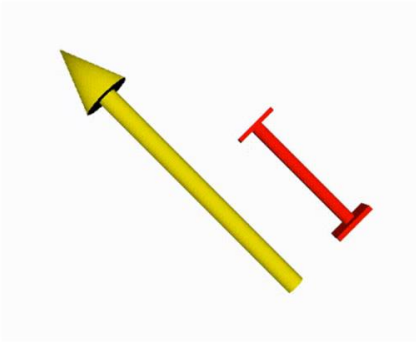
It has a length

It also has length.

Rigging with Triangles

Vectors

Normalize



Normalizing a vector changes length to ‘1’ but still points in the same direction, which makes many calcluations much easier.

Rigging with Triangles

Vectors

Can Represent a Location

Relative to something else

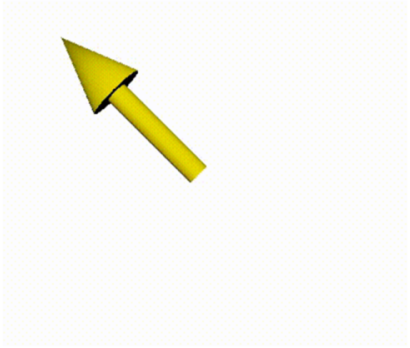
But also can be used to represent a location relative to something else, including origin.

Rigging with Triangles

Reversing a Vector

Vector * -1.0

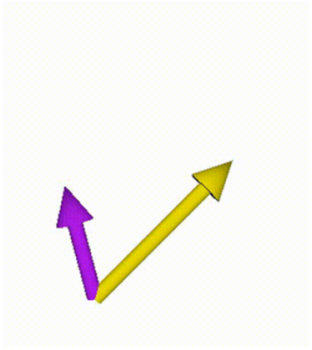
[1,2,3] * -1.0 = [-1,-2,-3]



Reversing a vector is pretty easy: We multiply the vector by -1, which in turn just multiplies each element of the vector by -1, and the resulting vector points in the opposite direction.

Rigging with Triangles

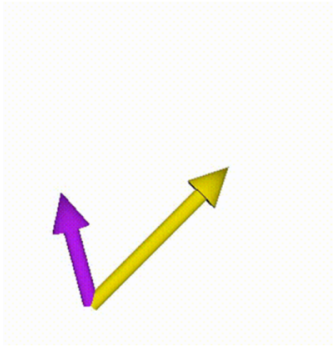
Adding Vectors



Adding two vectors gives you a third vector. This is done simply by adding the indices of each vector in turn. That vector's direction is the average of the direction of the operands. Dividing the result by two gives you the average of the two operands, both in direction and length.

Rigging with Triangles

Subtracting Vectors



Subtracting two vectors is similar, we subtract the indices of each vector to produce a third vector.* The result is a mirror of the second vector across the first vector.*

Rigging with Triangles

Finding a Vector between two points

Destination - Origin

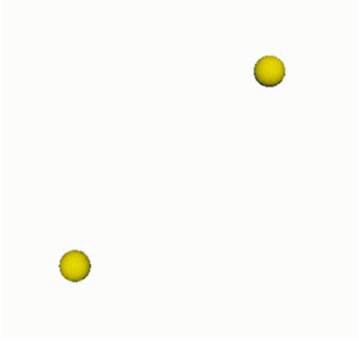
When we represent points in space as vectors, we can use subtraction to find a vector between two points. The destination point minus the origin point.

Rigging with Triangles

Finding a Vector Between 2 Points

[5,4,3] - [2,3,1] = [3,1,2]

5 - 2 = 3
4 - 3 = 1
3 - 1 = 2



We want the vector between these two points. The “destination” point is at [5,4,3].* The “Origin” point is at [2,3,1].* The resulting vector that aims from 2,3,1 to 5,4,3 is 3,1,2.*

Rigging with Triangles

Vector Length

$$\|\mathbf{a}\| = \sqrt{{a_1}^2 + {a_2}^2 + {a_3}^2}$$

The length of a vector can be calculated with the Pythagorean Theorem.

Rigging with Triangles

Distance between two Vectors

$$d(p,q) = \sqrt{(p_1 - q_1)^2 + (p_2 - q_2)^2 + (p_3 - q_3)^2}.$$

The distance between two points, can similarly be computed with this “Euclidean Distance” formula. Inside the parentheses, we’re just subtracting the components of each vector.

Rigging with Triangles

The Dot Product

The DOT product is one of the tools we use on vectors.

Rigging with Triangles

The Dot Product

Definition

$$a \cdot b = (ax * bx) + (ay * by) + (az * bz)$$

The math behind it is pretty simple: it’s the x’s, y’s, and z’s of two vectors multiplied, then added together

Rigging with Triangles

The Dot Product

The dot product of 2 *normalized* vectors returns a float (*scalar*) between -1 and +1

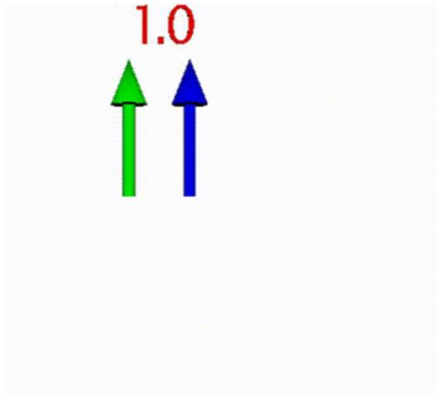
$$[a,b,c] \cdot [d,e,f] = g$$

The dot product of two normalized vectors returns a “scalar” value, or float value between -1 and 1.

Rigging with Triangles

The Dot Product

same direction: 1.0
opposite direction: -1.0
perpendicular: 0.0

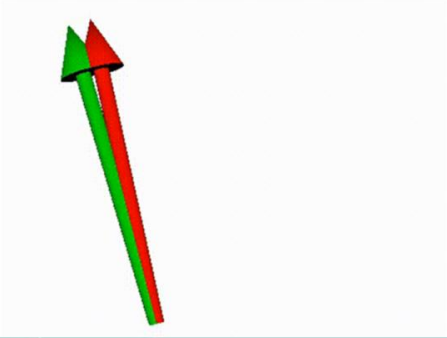


The dot product of two vectors pointing in the same direction is 1, pointing in opposite direction is -1, and two perpendicular vectors' dot product is zero. There are values in between as well: the dot product of vectors gives a similar result as the cosine of scalars.

Rigging with Triangles

The Dot Product

acos(v1 dot v2) = angle between vectors in radians



The ArcCosine of of the dot product of two normalized vectors is the ANGLE between the vectors in radians.

Rigging with Triangles

The Cross Product

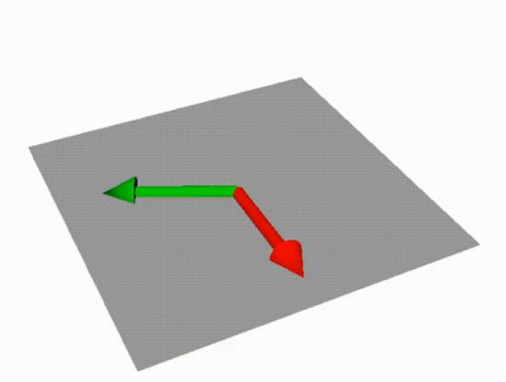
Where DOT products result in a single scalar value, the CROSS product of two vectors returns a third vector.

Rigging with Triangles

The Cross Product

- The Cross Product of 2 vectors returns a third vector

- The normal of the plane they define



That vector is the “normal” of the plane they define.

Rigging with Triangles

The Cross Product

Definition

$$c.x = a.y*b.z - a.z*b.y$$
$$c.y = a.z*b.x - a.x*b.z$$
$$c.z = a.x*b.y - a.y*b.x$$

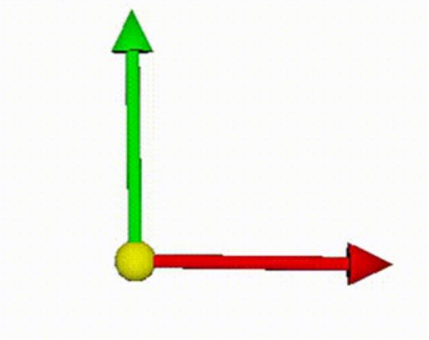
The math behind the cross product isn’t really too complicated either: It involves multiplying and subtracting elements of the two input vectors.

Rigging with Triangles

The Cross Product

"right-handed" coord system:

- Vector1 goes Right (X)
- Vector2 goes Up (Y)
- The cross product (Z) will come AT you

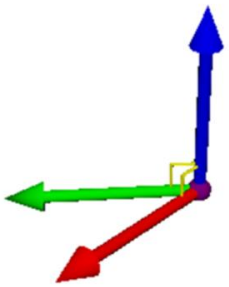


Maya uses a "right-handed" coordinate system: if vector1 goes right and vector2 goes up, the cross product will come AT you. 3d studio max is also right-handed, but since it's Z-Up, X points to the right, Y points away from you, the cross product Z will then point up

Rigging with Triangles

The Cross Product

The **dot product** against both source vectors will be 0



The dot product of the resulting vector against both source vectors will be 0, because it will be 90 degrees from either of the two input values.

Rigging with Triangles

Transformation Matrix

Finally, we come to the Transformation Matrix.

Rigging with Triangles

Transformation Matrix

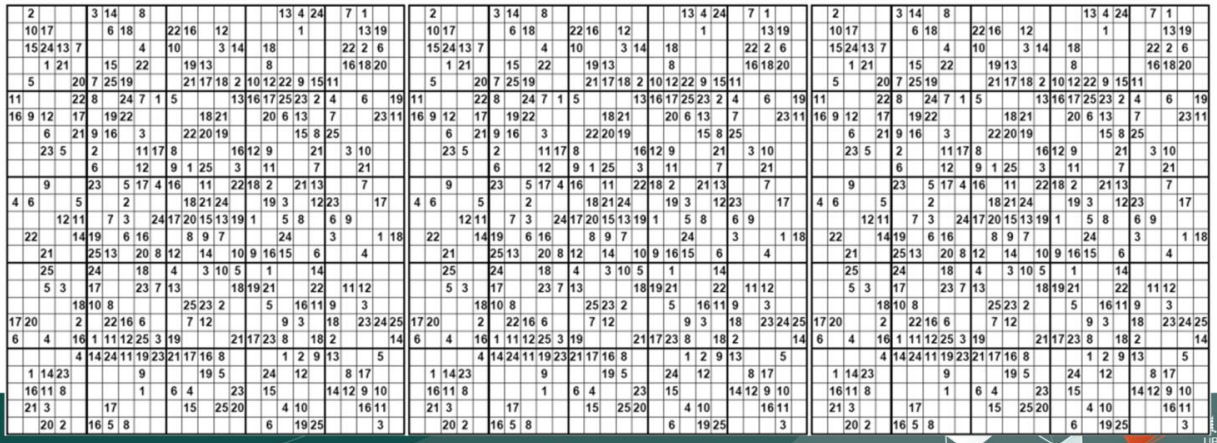
Scary 4x4 grid of 16 values

$$\begin{pmatrix} m_0 & m_4 & m_8 & m_{12} \\ m_1 & m_5 & m_9 & m_{13} \\ m_2 & m_6 & m_{10} & m_{14} \\ m_3 & m_7 & m_{11} & m_{15} \end{pmatrix}$$

It’s a scary 4x4 grid of 16 values.

Rigging with Triangles

Transformation Matrix



This is how that matrix used to look to me before I understood it.

Rigging with Triangles

Transformation Matrix

Maya will return a matrix as a flat list of 16 numbers

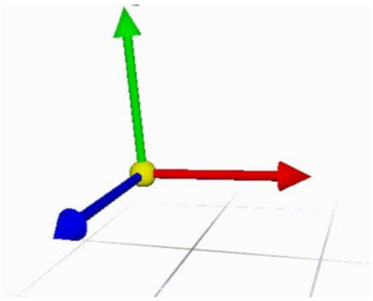
```
print cmds.getAttr('pSphere1.matrix')
[1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0, 0.0, 0.0, 0.0, 0.0, 1.0]
```

Maya will return a matrix as a flat list of 16 numbers, which is just as intimidating.

Rigging with Triangles

Transformation Matrix

X axis	1.0	0.0	0.0	0.0
Y axis	0.0	1.0	0.0	0.0
Z axis	0.0	0.0	1.0	0.0
Translation	-2.0	2.0	0.0	1.0

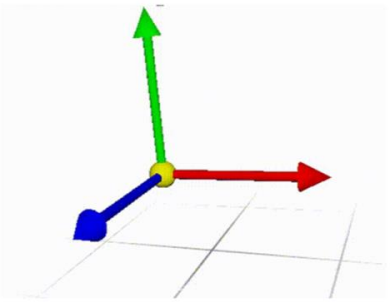


But what it really is: 4 vectors: The X Axis

Rigging with Triangles

Transformation Matrix

X axis	1.0	0.0	0.0	0.0
Y axis	0.0	1.0	0.0	0.0
Z axis	0.0	0.0	1.0	0.0
Translation	-2.0	2.0	0.0	1.0

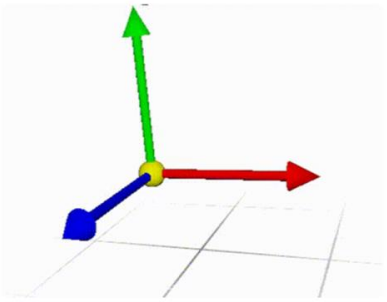


The Y Axis

Rigging with Triangles

Transformation Matrix

X axis	1.0	0.0	0.0	0.0
Y axis	0.0	1.0	0.0	0.0
Z axis	0.0	0.0	1.0	0.0
Translation	-2.0	2.0	0.0	1.0

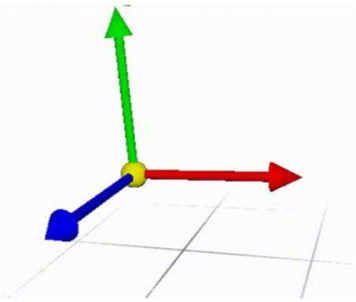


the Z Axis

Rigging with Triangles

Transformation Matrix

X axis	1.0	0.0	0.0	0.0
Y axis	0.0	1.0	0.0	0.0
Z axis	0.0	0.0	1.0	0.0
Translation	-2.0	2.0	0.0	1.0



and the fourth row is the translation as a Vector. So that red, green, and blue transform manipulator we're all familiar with is just a visual representation of a transformation matrix.

Rigging with Triangles

Transformation Matrix

Identity Matrix

[1,0,0] [0,1,0] [0,0,1] [0,0,0]

The term “Identity Matrix” refers to a matrix where the values are aligned to the world

Rigging with Triangles

Transformation Matrix

Identity Matrix

X vector points in positive X world space
Y vector points in positive Y world space
Z vector points in positive Z world space
translation is 0,0,0

The first vector points toward positive X world space, the Y vector points in positive Y world space, Z vector points in positive Z world space, and the translation is at the world origin, or [0,0,0].

Rigging with Triangles

Transformation Matrix

Fourth value in each row is 0

Except “transform matrix” translate row fourth element is 1

Fourth value in each row is 0, except the translation row, the fourth element of which is 1.0.

Rigging with Triangles

Transformation Matrix

Fourth value in each row is 0

Except “transform matrix” translate row fourth element is 1

4th column is a “hack” to allow translation to be represented in a matrix

https://stackoverflow.com/questions/2465116/understanding-opengl-matrices/2465290#2465290

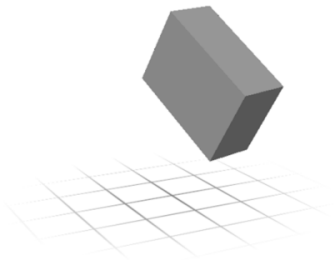
The 4th column is a “hack” to allow translation to be represented in a matrix.

Rigging with Triangles

Transformation Matrix

In world space:

Orientation, Scale, and Position from origin

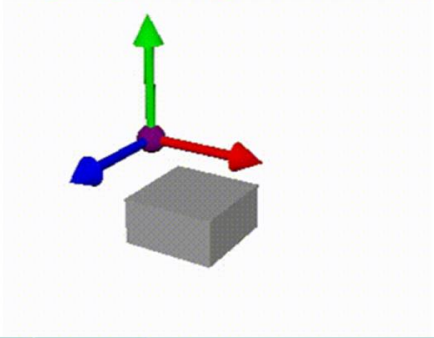


A Transformation Matrix In World Space describes an object's orientation, scale, and position from origin.

Rigging with Triangles

Transformation Matrix

Scale: changing lengths on x,y,z vectors

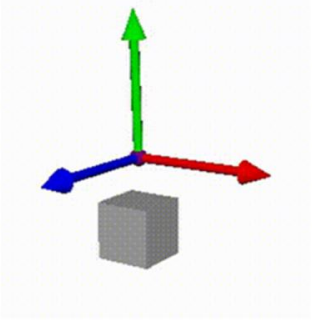


We can alter the Scale of the object by changing lengths on x,y,z vectors.

Rigging with Triangles

Transformation Matrix

Skew: making x,y,z vectors non-orthogonal



We can Skew the object by making x,y,z vectors non-orthogonal to each other.

Rigging with Triangles

Matrix Multiplication

We can multiply Matrices.

Rigging with Triangles

Matrix Multiplication

Non-commutative

(AM*BM != BM*AM)

$$\begin{bmatrix} 3 & 4 \\ 1 & 2 \end{bmatrix} \times \begin{bmatrix} 3 & 4 \\ 5 & 6 \end{bmatrix}$$

Matrix multiplication is non-commutative, in other words A times B doesn't necessarily equal B times A because the math involves multiplying the rows of the first matrix with the columns of the second.

Rigging with Triangles

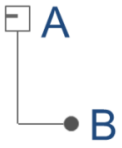
Inverse matrix

A.inverse() * A = [1,0,0] [0,1,0] [0,0,1] [0,0,0] = “identity"

The Inverse of a matrix is the transformation needed to multiply against a matrix in order to get an identity result, so the axes are aligned to the world and the translation is 0.

Rigging with Triangles

Inverse matrix



B.localMatrix == A.worldMatrix.inverse() * B.worldMatrix

If B is the child of A, we can multiply the inverse of A’s world matrix with B’s world matrix to get the local transformation matrix of B.

Rigging with Triangles

Operation on object in local space

Perform an operation

Move back to original space

```
A = object.worldMatrix
P = object.parentWorldMatrix
M = P.inverse() * A
move( M, [1, 0.5, 1] )
object.worldMatrix = P * M
```

We can also change an object's space to world or another object's local space, perform an operation, and return it to its original space.

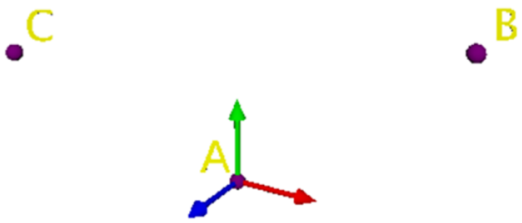
Rigging with Triangles

Applications in Rigging

Let’s build a matrix.

Rigging with Triangles

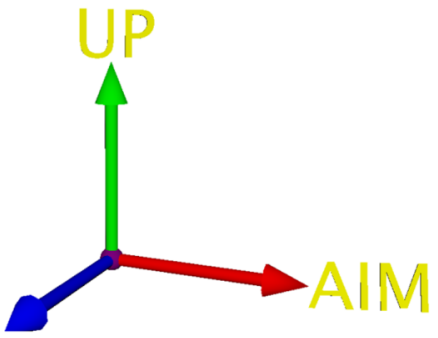
Aiming node A at node B



Here's some points in space, A,B, and C. A is currently aligned to the world, but we want to aim A at B, and use C as the upvector

Rigging with Triangles

Aiming node A at node B



A's X axis will be the axis we aim at B, and its Y axis will be point as much toward C as possible.

Rigging with Triangles

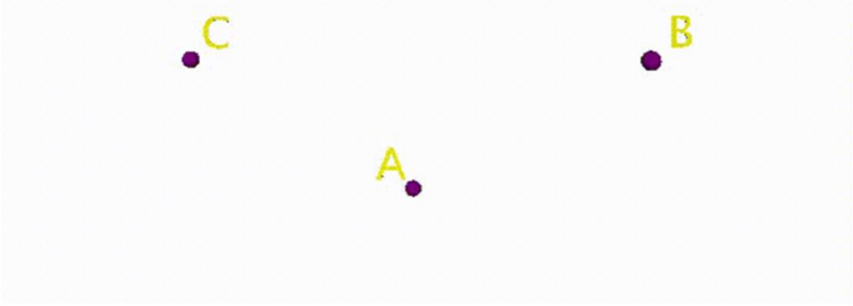
Aiming node A at node B

First we get the world position for A,B, and C.

Rigging with Triangles

Aiming node A at node B

aim_X = [bx,by,bz] - [ax,ay,az]

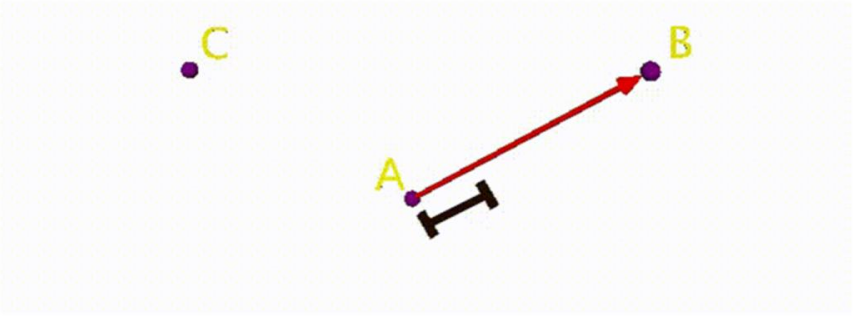


We know how to find the vector from A to B: it's B minus A.

Rigging with Triangles

Aiming node A at node B

aim_X = normalize(aim_X)

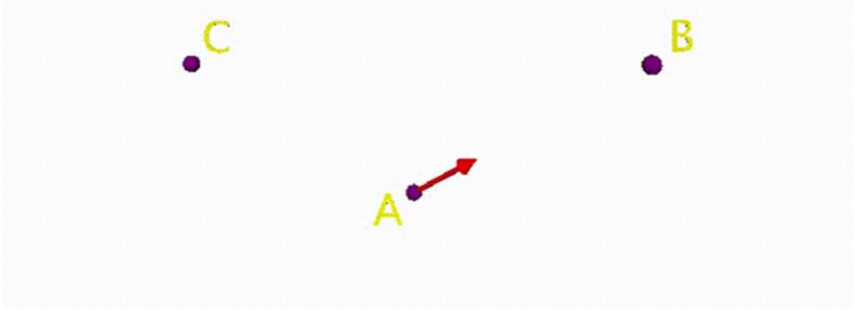


I don’t know how long that vector is, but we’ll normalize it so its length is 1, but it’s still pointing from A toward B.

Rigging with Triangles

Aiming node A at node B

aim_Y = [cx,cy,cz] - [ax,ay,az]

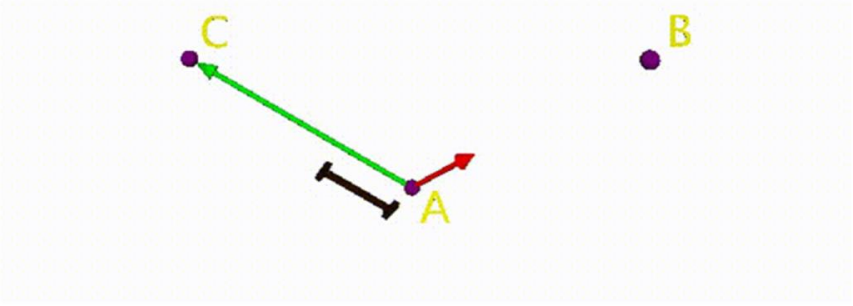


We get the second vector the same way, C minus A.

Rigging with Triangles

Aiming node A at node B

aim_Y = normalize(aim_Y)

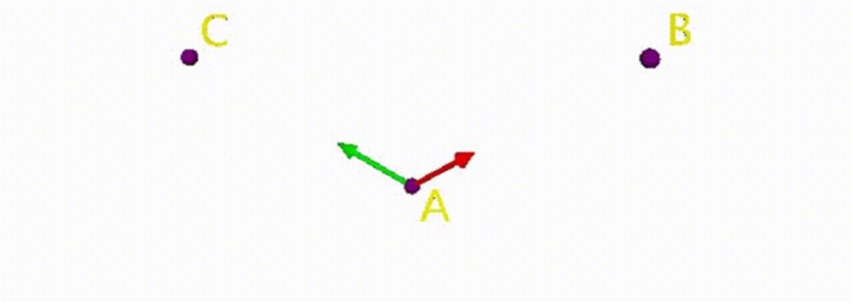


And we normalize that vector also.

Rigging with Triangles

Aiming node A at node B

aim_Z = aim_X X aim_Y

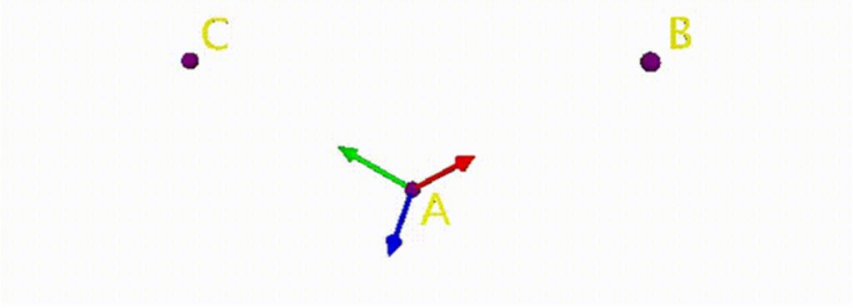


Here's the fun part: we get the third vector by calculating the cross product of the first two vectors.

Rigging with Triangles

Adjusting the Up Vector

aim_Y = aim_Z X aim_X



Now aim_x and aim_z are 90 degrees from each other, and aim_Y and aim_Z are 90 degrees from each other, aim_x and aim_y aren't necessarily 90 degrees from each other. So we get an adjusted aim_Y by crossing aim_Z and aim_X.

Rigging with Triangles

Aiming node A at node B

M = [aim_X, aim_Y, aim_Z, A_pos]

We make a new transformation matrix “M” by combining those three aim vectors and the world position of A.

Rigging with Triangles

Aiming node A at node B

M = [aim_X, aim_Y, aim_Z, A_pos]

aim_X.x	aim_X.y	aim_X.z	0
aim_Y.x	aim_Y.y	aim_Y.z	0
aim_Z.x	aim_Z.y	aim_Z.z	0
A.pos_x	A.pos_y	A.pos_z	1

This is what the matrix looks like broken down into individual float values: the X, Y, and Z elements of each of the X, Y, Z, and translation vectors.

Rigging with Triangles

Aiming node A at node B

A.worldMatrix = M

Finally we set the worldmatrix of A to the matrix we constructed.

Rigging with Triangles

Aiming node A at node B

If P is the parent of A:

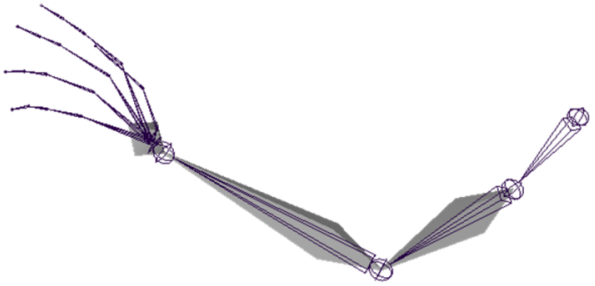
A.matrix = P.worldmatrix.inverse() * M

If A had a parent P, we would set A’s local matrix to the inverse of P’s world matrix times M.

Rigging with Triangles

Arm

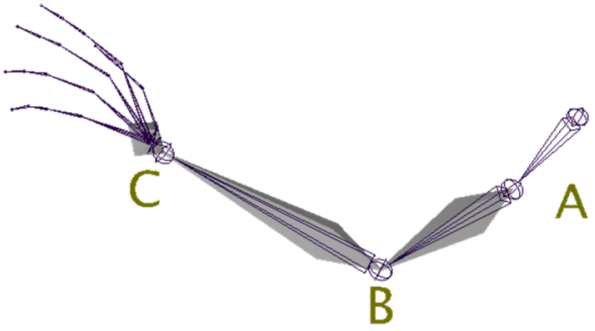
Shoulder Elbow Wrist



Here’s a pretty common operation, calculating the correct position of an arm’s pole vector control. Here’s an arm.

Rigging with Triangles

Arm

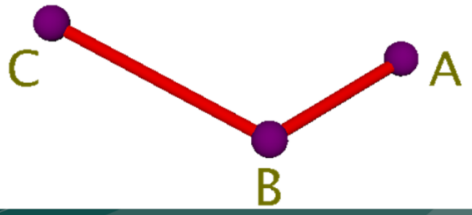


we'll call the shoulder point A, the elbow point B, and the wrist point C.

Rigging with Triangles

Arm

Shoulder Elbow Wrist



AB is the upper arm, and BC is the forearm.

Rigging with Triangles

Pole Vector

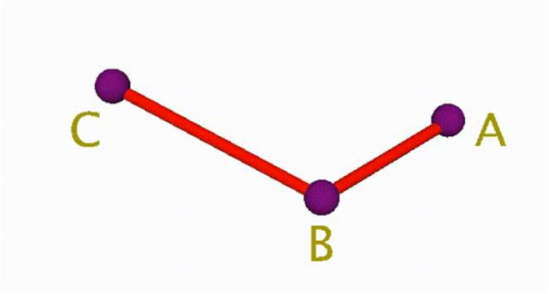
Get Shoulder-Wrist vector

Project elbow to shldr-wrist with dot product

New vector = normalize(elbow - elbow_proj)

Length = (shldr-elbow + elbow-wrist) * 0.5

Elbow pos + (length * newvector)



First, let's get the shoulder-wrist vector by subtracting: C minus A. Hey look, there's a Triangle!

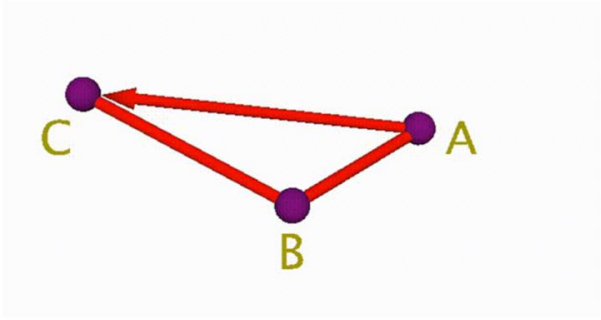
Rigging with Triangles

Pole Vector

Get Shoulder-Wrist vector

Project elbow to shldr-wrist with dot product

```
ab = b-a
ac = c-a
p = a + ((ab · ac)/(ac · ac)) * ac
```



This part is a little tricky: We're going to project elbow to the shldr-wrist vector using some dot product math. That gives us point P along vector AC.

Rigging with Triangles

Pole Vector

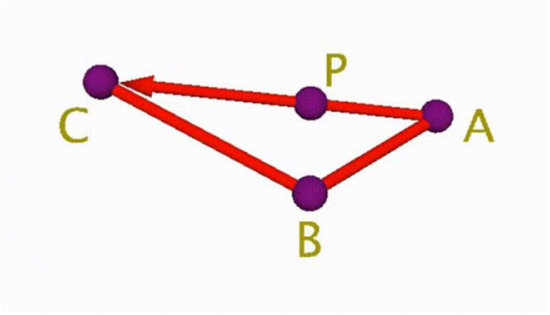
Get Shoulder-Wrist vector

Project elbow to shldr-wrist with dot product

New vector = normalize(elbow - elbow_proj)

Length = (shldr-elbow + elbow-wrist) * 0.5

Elbow pos + (length * newvector)



We make a new vector by subtracting the elbow position B minus P, and we normalize that vector.

Rigging with Triangles

Pole Vector

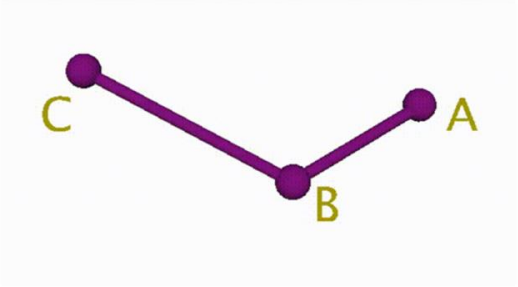
Get Shoulder-Wrist vector

Project elbow to shldr-wrist with dot product

New vector = normalize(elbow - elbow_proj)

Length = (shldr-elbow + elbow-wrist) * 0.5

Elbow pos + (length * newvector)



It’s personal preference how far off the elbow we position the pole vector, but my rule of thumb is half of the length of the entire arm, or AB plus BC times 0.5.

Rigging with Triangles

Pole Vector

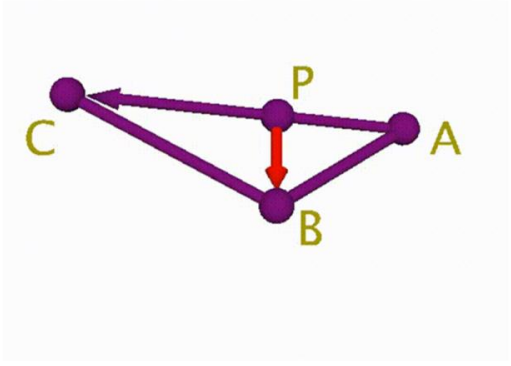
Get Shoulder-Wrist vector

Project elbow to shldr-wrist with dot product

New vector = normalize(elbow - elbow_proj)

Length = (shldr-elbow + elbow-wrist) * 0.5

Elbow pos + (length * newvector)

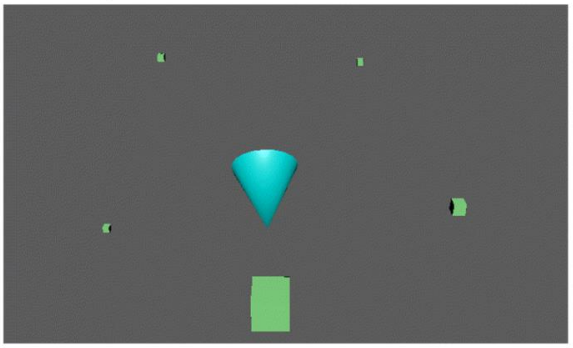


Finally, we set the position of the pole vector control: multiply the normalized PB vector times the length value we calculated, and add that to the elbow position B.

Rigging with Triangles

PSD node

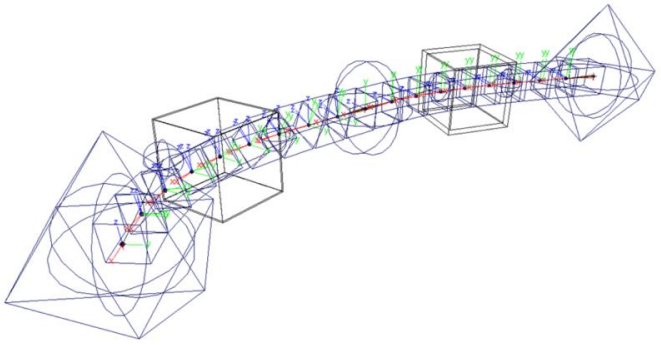
Using the dot product to get a value of how much one node aims at another



Here's a quick demo of a Pose Space node I wrote using the dot product to calculate a value indicating how much a driver object points toward a list of other objects. Remember, the dot product of two vectors pointing in the same direction is 1, so I'm just comparing one of the driver node's axes with the normalized vector of the target node's position minus the driver's position.

Rigging with Triangles

Building on the Basics



I've given a few examples, but once you start becoming comfortable with vectors and matrices, you can build larger and more complicated systems under the hood of your animation rigs or whatever else you need mathematical solutions for.

Rigging with Triangles

Thanks

That’s all I have, thanks.

Rigging with Triangles

Questions

?

Any Questions

Rigging with Triangles

Appendices

Rigging with Triangles

Distance and Length

```
import math
import numpy.linalg as la
import maya.OpenMaya as OpenMaya

v = [1,2,3]

math.sqrt( (v[0]**2) + (v[1]**2) + (v[2]**2) )

la.norm(v)

OpenMaya.MVector(*v).length()
```

Here's a few ways to find the length of a vector in Python:

1) Using the math module, and the pythagorean formula
2) In numpy, the "linear algebra" module's "norm" function
3) With an In OpenMaya MVector's length() function

Rigging with Triangles

The Dot Product

dot product in numpy

```
import numpy as np

normalize = lambda a: np.array(a) / np.linalg.norm(a)

v1 = [-3.132, -4.317, 6.369]
v2 = [-6.032, 5.884, 1.618]
n1 = normalize(v1)
n2 = normalize(v2)
n1.dot(n2)
```

2 vectors, normalize them, get dot product

Rigging with Triangles

The Cross Product

cross product in numpy

```
import numpy as np
import numpy.linalg as la

normalize = lambda a: np.array(a) / la.norm(a)

v1 = [-3.132, -4.317, 6.369]
v2 = [-6.032, 5.884, 1.618]
n1 = normalize(v1)
n2 = normalize(v2)
numpy.cross(n1,n2)
```


cross prod in numpy
import numpy as np
import numpy.linalg as la

normalize = lambda a: np.array(a) / la.norm(a)

v1 = [-3.132, -4.317, 6.369]
v2 = [-6.032, 5.884, 1.618]
n1 = normalize(v1)
n2 = normalize(v2)
numpy.cross(n1,n2)

Web resources for learning more about vectors and manipulating them…

Khan Academy

https://www.khanacademy.org/math/linear-algebra/vectors-and-spaces/vectors/v/vector-introduction-linear-algebra

WolframAlpha

http://www.wolframalpha.com/examples/Algebra.html