



GAME DEVELOPMENT > PROGRAMMING

Let's Build a 3D Graphics Engine: Linear Transformations

Advertisement

by [Kyle Sloka-Frey](#) 22 May 2013Difficulty: Intermediate Length: Medium Languages: English ▼

Programming

Platform Agnostic

HTML5



This post is part of a series called [Let's Build a 3D Graphics Software Engine](#).

◀ [Let's Build a 3D Graphics Engine: Points, Vectors, and Basic Concepts](#)

▶ [Let's Build a 3D Graphics Engine: Spaces and Culling](#)

Welcome to the second part of our 3D Graphics Engine series! This time we are going to be talking about *linear transformations*, which will let us alter properties like the rotation and scaling of our vectors, and look at how to apply them to the classes we've already built.

If you haven't already read [the first part of this series](#), I suggest you do so now. Just in case you don't remember, here is a quick recap of what we created last time:

```
01 Point Class
02 {
03     Variables:
04         num tuple[3]; //(x,y,z)
05
06     Operators:
07         Point AddVectorToPoint(Vector);
08         Point SubtractVectorFromPoint(Vector);
09         SubtractPointFromPoint(Point);
10
11     Functions:
12         //draw a point at its position tuple with your favorite graphics API
13         drawPoint;
14 }
15
16 Vector Class
17 {
18     Variables:
19         num tuple[3]; //(x,y,z)
20
21     Operators:
22         Vector AddVectorToVector(Vector);
23         Vector SubtractVectorFromVector(Vector);
24 }
```

Those two classes will be the basis of our entire graphics engine, where the first represents a point (a physical location within your space) and the second represents a vector (the space/movement between two points).

For our discussion on linear transformations, you should make a small change to the Point class: instead of outputting data to a console line like before, use your favorite Graphics API and have the function draw the current point to the screen.

Foundations of Linear Transformations

Just a warning: Linear Transformation equations look a lot worse than they actually are. There will be some trigonometry involved, but you don't have to actually know *how* to do that trigonometry: I will explain what you have to give each function and what you will get out, and for the in-between stuff you can just use any calculator or math library that you may have.

Tip: If you do want to have a better grasp of the inner workings of these equations, then you should [watch this video](#) and [read this PDF](#).

All linear transformations take this form:

$$B = F(A)$$

This states that if you have a linear transformation function $F()$, and your input is the vector A , then your output will be the vector B .

Each of these pieces - the two vectors and the function - can be represented as a matrix: the vector B as a 1x3 matrix, the vector A as another 1x3 matrix, and the linear transformation F as a 3x3 matrix (a *transformation matrix*).

This means that, when you expand the equation, it looks like this:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} f_{00} & f_{01} & f_{02} \\ f_{10} & f_{11} & f_{12} \\ f_{20} & f_{21} & f_{22} \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix}$$

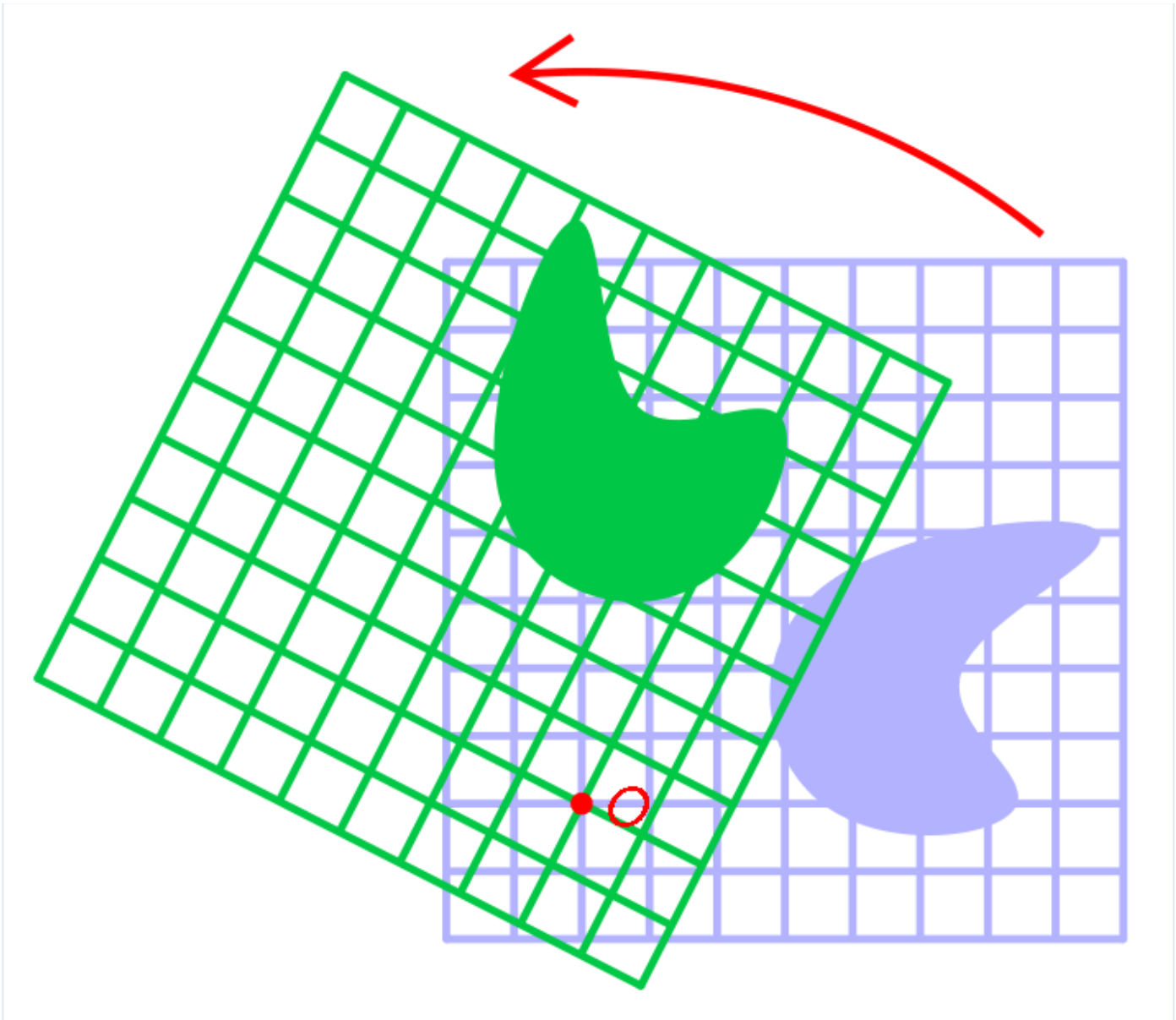
If you have ever taken a class in trigonometry or linear algebra, you are probably starting to remember the nightmare that was matrix math. Luckily, there is a simpler way to write out this equation to take most of the trouble out of it. It looks like this:

$$\begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} = \begin{bmatrix} f_{00}a_0 + f_{01}a_1 + f_{02}a_2 \\ f_{10}a_0 + f_{11}a_1 + f_{12}a_2 \\ f_{20}a_0 + f_{21}a_1 + f_{22}a_2 \end{bmatrix}$$

However, these equations may be altered by having a second input, such as in the case of rotations, where a vector and its rotation amount must both be given. Let's take a look at how rotations work.

Rotations

A rotation is, by definition, a circular movement of an object around a point of rotation. The point of rotation for our space can be one of three possibilities: either the XY plane, the XZ plane, or the YZ plane (where each plane is made up of two of our basis vectors that we discussed in the first part of the series).



Our three points of rotation mean that we have three separate rotation matrices, as follows:

XY rotation matrix:

$$\begin{bmatrix} \cos\theta & -\sin\theta & 0 \\ \sin\theta & \cos\theta & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

XZ rotation matrix:

$$\begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix}$$

YZ rotation matrix:

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\theta & -\sin\theta \\ 0 & \sin\theta & \cos\theta \end{bmatrix}$$

So to rotate a point A around the XY plane by 90 degrees ($\pi/2$ radians - most math libraries have a function for converting degrees into radians), you would follow these steps:

$$\begin{aligned} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} &= \begin{bmatrix} \cos\frac{\pi}{2} & -\sin\frac{\pi}{2} & 0 \\ \sin\frac{\pi}{2} & \cos\frac{\pi}{2} & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \\ &= \begin{bmatrix} \cos\frac{\pi}{2}a_0 + -\sin\frac{\pi}{2}a_1 + 0a_2 \\ \sin\frac{\pi}{2}a_0 + \cos\frac{\pi}{2}a_1 + 0a_2 \\ 0a_0 + 0a_1 + 1a_2 \end{bmatrix} \\ &= \begin{bmatrix} 0a_0 + -1a_1 + 0a_2 \\ 1a_0 + 0a_1 + 0a_2 \\ 0a_0 + 0a_1 + 1a_2 \end{bmatrix} \\ &= \begin{bmatrix} -a_1 \\ a_0 \\ a_2 \end{bmatrix} \end{aligned}$$

So if your initial point A was $(3, 4, 5)$, then your output point B would be $(-4, 3, 5)$.

Advertisement

Exercise: Rotation Functions

As an exercise, try creating three new functions for the `Vector` class. One should rotate the vector around the XY plane, one around the YZ plane, and one around the XZ plane. Your functions should receive the desired amount of degrees for rotation as an input, and return a vector as an output.

The basic flow of your functions should be as follows:

1. Create output vector.
2. Convert the degree input into radian form.
3. Solve for each piece of the output vectors tuple by using the equations above.
4. Return the output vector.

Scaling

Scaling is a transformation that either enlarges or diminishes an object based on a set scale.

Performing this transformation is fairly simple (at least compared to rotations). A scaling transformation requires two inputs: an *input vector* and a *scaling 3-tuple*, which defines how the input vector should be scaled in regards to each of the space's basis axes.

For example, in the scaling tuple (s_0, s_1, s_2) , s_0 represents the scaling along the X axis, s_1 along the Y axis, and s_2 along the Z axis.

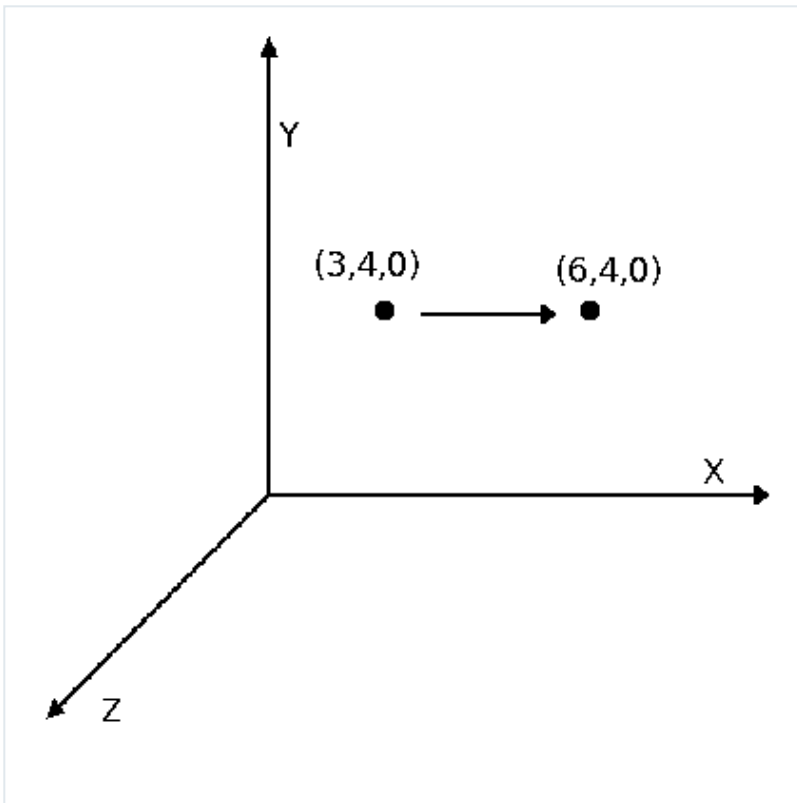
The scaling transformation matrix is as follows (where s_0 , s_1 , and s_2 are the elements of the scaling 3-tuple):

$$\begin{bmatrix} s_0 & 0 & 0 \\ 0 & s_1 & 0 \\ 0 & 0 & s_2 \end{bmatrix}$$

In order to make the input vector $A(a_0, a_1, a_2)$ twice as large along the X axis (that is, using a scaling 3-tuple $S = (2, 1, 1)$), the math would look like this:

$$\begin{aligned}
 \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} &= \begin{bmatrix} s0 & 0 & 0 \\ 0 & s1 & 0 \\ 0 & 0 & s2 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \\
 &= \begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \end{bmatrix} \\
 &= \begin{bmatrix} 2a_0 + 0a_1 + 0a_2 \\ 0a_0 + 1a_1 + 0a_2 \\ 0a_0 + 0a_1 + 1a_2 \end{bmatrix} \\
 &= \begin{bmatrix} 2a_0 \\ a_1 \\ a_2 \end{bmatrix}
 \end{aligned}$$

So if given the input vector $A = (3, 4, 0)$, then your output vector B would be $(6, 4, 0)$.



Exercise: Scaling Functions

As another exercise, add a new function to your vector class for scaling. This new function should take in a scaling 3-tuple and return an output vector.

The basic flow of your functions should be as follows:

1. Create output vector.
2. Solve for each piece of the output vectors tuple by using the equation above (which can be simplified to $y_0 = x_0 * s_0; y_1 = x_1 * s_1; y_2 = x_2 * s_2$).
3. Return the output vector.

Advertisement

Let's Build Something!

Now that you've got linear transformations under your belt, let's build a quick little program to show off your new skills. We're going to make a program that draws a group of points to the screen, and then allows us to modify them as a whole by performing linear transformations on them.

Before starting, we will also want to add another function to our `Point` class. This will be called `setPointToPoint()`, and will simply set the current point's position to that of the point that is passed to it. It will receive a point as an input, and will return nothing.

Here are some quick specifications for our program:

- The program will hold 100 points in an array.
- When the **D** key is pressed, the program will wipe the current screen and redraw the points.
- When the **A** key is pressed, the program will scale all of the points' locations by 0.5.

- When the **S** key is pressed, the program will scale all of the points' locations by 2.0.
- When the **R** key is pressed, the program will rotate all of the points' location by 15 degrees on the XY plane.
- When the **Escape** key is pressed, the program will exit (unless you're making it with JavaScript or another web-oriented language).

Our current classes:

```

01 Point Class
02 {
03     Variables:
04         num tuple[3]; //(x,y,z)
05
06     Operators:
07         Point AddVectorToPoint(Vector);
08         Point SubtractVectorFromPoint(Vector);
09         Vector SubtractPointFromPoint(Point);
10         //sets the current point's position to that of the inputted point
11         Null SetPointToPoint(Point);
12     Functions:
13         //draw a point at its position tuple with your favorite graphics API
14         drawPoint;
15 }
16
17 Vector Class
18 {
19     Variables:
20         num tuple[3]; //(x,y,z)
21
22     Operators:
23         Vector AddVectorToVector(Vector);
24         Vector SubtractVectorFromVector(Vector);
25         Vector RotateXY(degrees);
26         Vector RotateYZ(degrees);
27         Vector RotateXZ(degrees);
28         Vector Scale(s0,s1,s2);
29 }

```

With those specifications, let's look at what our code could be:

```

01 main{
02     //setup for your favorite Graphics API here
03     //setup for keyboard input (may not be required) here
04
05     //create an array of 100 points
06     Point Array pointArray[100];
07
08     for (int x = 0; x < pointArray.length; x++)
09     {
10         //Set its location to a random point on the screen

```

```
11     pointArray[x].tuple = [random(0,screenWidth), random(0,screenHeight), random(0,
12 }
13
14 //this function clears the screen and then draws all of the points
15 function redrawScreen()
16 {
17     //use your Graphics API's clear screen function
18     ClearTheScreen();
19
20     for (int x = 0; x < pointArray.length; x++)
21     {
22         //draw the current point to the screen
23         pointArray[x].drawPoint();
24     }
25 }
26
27 // while the escape is not being pressed, carry out the main loop
28 while (esc != pressed)
29 {
30     // perform various actions based on which key is pressed
31     if (key('d') == pressed)
32     {
33         redrawScreen();
34     }
35     if (key('a') == pressed)
36     {
37         //create the space's origin as a point
38         Point origin = new Point(0,0,0);
39
40         Vector tempVector;
41         for (int x = 0; x < pointArray.length; x++)
42         {
43             //store the current vector address for the point, and set the point
44             tempVector = pointArray[x].subtractPointFromPoint(origin);
45             //reset the point so that the scaled vector can be added
46             pointArray[x].setPointToPoint(origin);
47             //scale the vector and set the point to its new, scaled location
48             pointArray[x].addVectorToPoint(tempVector.scale(0.5,0.5,0.5));
49         }
50         redrawScreen();
51     }
52
53     if(key('s') == pressed)
54     {
55         //create the space's origin as a point
56         Point origin = new Point(0,0,0);
57
58         Vector tempVector;
59         for (int x = 0; x < pointArray.length; x++)
60         {
61             //store the current vector address for the point, and set the point
62             tempVector = pointArray[x].subtractPointFromPoint(origin);
63             //reset the point so that the scaled vector can be added
64             pointArray[x].setPointToPoint(origin);
65             //scale the vector and set the point to its new, scaled location
66             pointArray[x].addVectorToPoint(tempVector.scale(2.0,2.0,2.0));
67         }
```

```
68         redrawScreen();
69     }
70
71     if(key('r') == pressed)
72     {
73         //create the space's origin as a point
74         Point origin = new Point(0,0,0);
75         Vector tempVector;
76         for (int x = 0; x < pointArray.length; x++)
77         {
78             //store the current vector address for the point, and set the point
79             tempVector = pointArray[x].subtractPointFromPoint(origin);
80             //reset the point so that the scaled vector can be added
81             pointArray[x].setPointToPoint(origin);
82             //scale the vector and set the point to its new, scaled location
83             pointArray[x].addVectorToPoint(tempVector.rotateXY(15));
84         }
85         redrawScreen();
86     }
87 }
88 }
```

Now you should have a cool, little program to show off all of your new techniques! You can [check out my simple demo here](#).

Conclusion


While we certainly didn't cover every possible linear transformation available, our micro-engine is starting to take shape.

As always, there are some things that got left out of our engine for simplicity (namely shearing and reflections in this part). If you want to find out more about those two types of linear transformations, you can find out more about them on [Wikipedia](#) and its related links.

In the next part of this series, we will be covering different view spaces and how to cull objects that are outside of our view.

If you need extra help, head over to Envato Studio, where you can find plenty of fantastic [3D Design & Modeling services](#). These experienced providers can help you with a

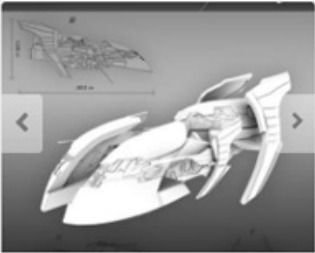
wide range of different projects, so just browse the providers, read the reviews and ratings, and choose the right person to help you.



3D Floor Plan Modeling and Rendering

Vertex-Design
100% Recommended


\$100
4 days



3D Object Modelling

WhiteX
100% Recommended

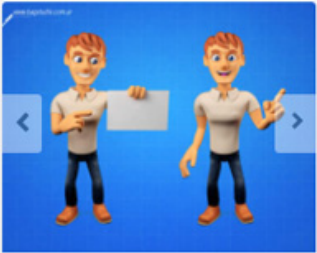
\$450
10 days



3D Logo Design

Gabey005
100% Recommended

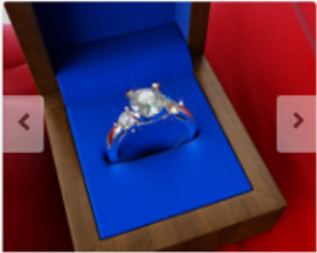
\$200
5 days



3D Character / Mascot Design

Gabey005
100% Recommended


\$300
7 days



Object Modelling / Rendering

meetai
100% Recommended

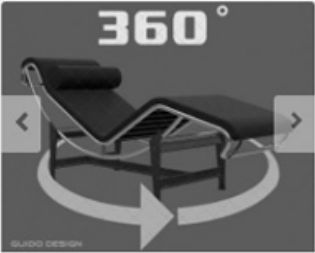
\$150
2 days



3d Rendering

unesdesign
100% Recommended


\$120
2 days



Turntable 3D Models

GuidoDesign
100% Recommended

\$100
2 days



Product Design, 3D Modeling & Visualization

xpshl
100% Recommended

\$250
4 days

3D Design & Modeling services on Envato Studio

Advertisement



Kyle Sloka-Frey

Data Scientist, Father, Game Dev, Developer, Futurist

Kyle is a data scientist, father, game dev, and all around analytics and organization lover. He enjoys space and all things Futurism.

 [kyleslokafrey](#)

 FEED  LIKE  FOLLOW

Weekly email summary

Subscribe below and we'll send you a weekly email summary of all new Game Development tutorials. Never miss out on learning about the next big thing.

Update me weekly

Translations

Envato Tuts+ tutorials are translated into other languages by our community members—you can be involved too!

Translate this post

Powered by



Advertisement

10 Comments

Gamedevtuts+



1 Login ▾

♥ Recommend 1

🐦 Tweet

f Share

Sort by Best ▾



Join the discussion...

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

**jdt** • 7 years ago

Great! Can't wait to read the next part!

2 ^ | ▾ • Reply • Share ›

**Marko** • 4 years ago

The graphics API used must support 3D graphics or can i use this on a 2D api. If i can use this on 2D then where can i find the piece of code that translates 3D coordinate onto 2D space.

1 ^ | ▾ • Reply • Share ›

**Milkev Gaming** • 5 years ago • edited

So, what can i use to run this code? does anyone know any online sites, or portable (it doesnt actually have to be, but that would be extremely usefull) programs i can use to run code?

^ | v • Reply • Share ›

**sk sam** • 5 years ago

wow its great

^ | v • Reply • Share ›

**sagar** • 6 years ago

what is math processing error here displaying every time...

^ | v • Reply • Share ›

**RCDMK** • 7 years ago

Just a note: in the demo, when you rotate the points the result also scales down the coordinates.

^ | v • Reply • Share ›

**silbano** → RCDMK • 7 years ago

This is a "bug". He is probably using direct casting of the trig functions value into an integer. This causes loss of accuracy, as casting to integer just drops the decimal part of the number. Instead, a round function should be used (in Java: Math.round()).

^ | v • Reply • Share ›

**paws101** → silbano • 6 years ago

Cant seem to fix it with any rounding, looking into it more.

Any clues on JS handing of float to int?

here is a code snipett of where the problem is

```
var radians = degrees * (Math.PI / 180.0); // was 180
this.x = this.x;
this.y = Math.cos(radians) * (this.y) -
Math.sin(radians) * (this.z); // tried math.round()
around most parts of this where float/int occurs
this.z = Math.sin(radians) * (this.y) +
Math.cos(radians) * (this.z);
```

^ | v 1 • Reply • Share ›

**starbeamrainbowlabs** • 7 years ago



Awesome series, but I don't understand the transformation matrices

QUICK LINKS - Explore popular categories

ENVATO TUTORIALS



JOIN OUR COMMUNITY



HELP



29,299

Tutorials

1,293

Courses

44,485

Translations

[Envato.com](#) [Our products](#) [Careers](#) [Sitemap](#)

© 2020 Envato Pty Ltd. Trademarks and brands are the property of their respective owners.

Follow Envato Tuts+   

Facebook

Twitter

Pinterest