

A Method for Head and Eye Aiming using Steepest Descent

BY SCOTT M. JOHNSON

Abstract

A solution is presented to aim an animated character's eyes at a target point. Optionally, the character can look directly at a target point by turning the character's neck. This is a good solution because visual results are good, and there is an accompanying XNA project (based off the XNA Skinning Sample) that implements the solution so that you can try it and evaluate it before you decide to add it to your title. Of course since the solution is in XNA it is implemented in C#. This problem has also been called Eye and Head Tracking but "aiming" seemed more appropriate.

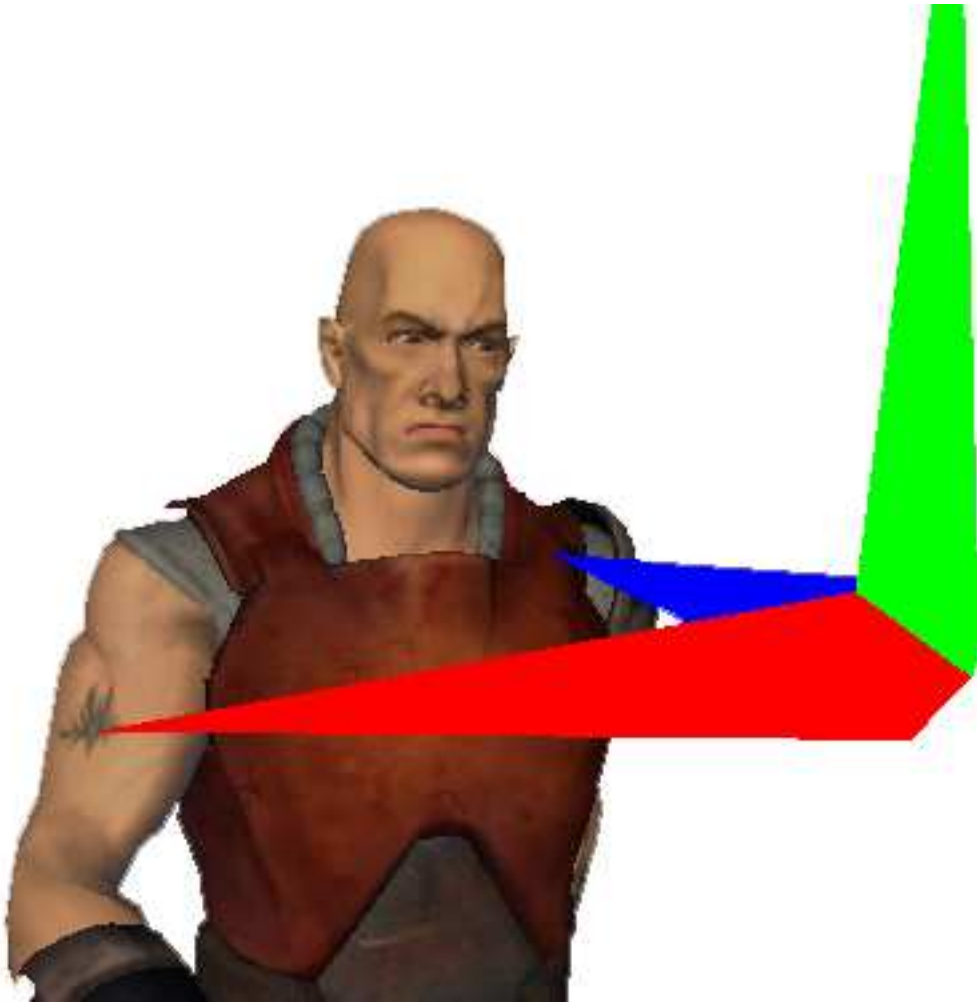


Figure 1. Thug looking at origin of target frame.

1 Problem

After you have animated characters in your game it is natural that you will then want them to look at or pay attention to items.

Pointing the eyes directly at a target by themselves is simple to do but deciding how to make several joints work together to accomplish the same thing puts you into the realm of Robotics and Inverse Kinematics (IK).

Specifically, the problem is to make an animated character's eyes point at a target position in world space. His joints should smoothly move into position over time and not exceed their range of motion. The solution must interface with an animation system so that the movement of the joints adds onto the movement of the character instead of completely replacing it. It is desired to keep the animator's work intact as much as possible while still accomplishing the goal of aiming the eyes. Lastly, there are times when a human's head is facing one direction and the eyes are looking in another. When something is particularly interesting, a human moves his head to look directly at it without turning his eyes away. This is a nice feature to have in a good solution.

There is one term that needs to be defined. In Robotics, there is always a chain of joints that ends in an effector. The effector is generally a tool like a paint nozzle or some sort of sensor that needs to be moved and oriented in a certain way. Our effector is represented by a unit vector coming from the left eye. The right eye will be aimed in the same direction in order to save a little computation but it is not hard to do a separate eye calculation for the right eye if needed. If the neck needs to be moved directly, then it will be considered the effector but mostly the discussion will focus on the left eye being the effector.

Technically stated, the problem is to orient the effector by modifying the joint matrices from the animation system so that the effector points at the target position. This solution will use three joints from the XNA Skinning Sample.

2 Math and Notation Review

2.1 XNA Coordinate Systems

XNA uses a right handed world coordinate system. To a character like the walking thug in the XNA Skinning Sample, +X is to his right, +Y is straight up, and +Z points behind him. He faces the -Z direction. Figure 1 shows this. There is a red, green, and blue reference frame floating in front of the character. For beginners, red is X, green is Y, and Z is blue. The joints (or bones) of the sample animation use another coordinate system. The bones use +X is up, +Y is forward, and +Z is to the left. The mathematics in this paper is tied to these systems.

2.2 Matrix Notation

Like Direct X, XNA uses row matrices to represent transforms between the spaces of joints. For reference, equation (1), (2) and (3) are the 4x4 homogenous row matrices used to rotate about each axis.

$$R_{x4}(x) = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(x) & \sin(x) & 0 \\ 0 & -\sin(x) & \cos(x) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (1)$$

$$R_{y4}(y) = \begin{pmatrix} \cos(y) & 0 & -\sin(y) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(y) & 0 & \cos(y) & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2)$$

$$R_{z4}(z) = \begin{pmatrix} \cos(z) & \sin(z) & 0 & 0 \\ -\sin(z) & \cos(z) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{pmatrix}. \quad (3)$$

A rotation of 90 degrees about the X axis is then $Rx4(\frac{\pi}{2})$. In this notation, R stands for Row matrix and a column matrix would start with C but only rows are used here. It is useful to have these written down and labeled as using rows to avoid confusing them with the same matrices found in textbooks that are tacitly column matrices.

A complete chain of matrices from the effector to world would look like this in row notation.

$$\vec{P}_{\text{world}} = \vec{P}_{\text{effector}} M_{\text{effector} \rightarrow \text{neck}} M_{\text{neck} \rightarrow \text{spine3}} M_{\text{spine3} \rightarrow \text{world}}$$

\vec{P}_{world} is just some 3D point in world space. The M's are 4x4 transform matrices which transform a vertex from its original space to a new space. The world space is at the far right in the equation and the spaces progress from world to the most distant from right to left.

3 Solution

This solution uses three bones of the sample animation in order to orient the effector. The left eye is always pointed directly at the target if it can reach, then the neck is oriented, and then the bone called spine3 is used. The effect of spine3 was minimal but was it was used anyway to show that the solution is general to N joints.

Of course orienting a joint in the chain changes the orientation of joints later in the chain. So the problem is to find a way to orient a local joint that will orient the effector in the correct way. There is a simple homogenous 4x4 transform between any joint that can be changed and the effector. This matrix will be called the effectorToJoint matrix. So if spine3 was being oriented, effectorToJoint would be the product of all the matrices in the chain to its left (shown below in paranthesis).

$$\vec{P}_{\text{world}} = \vec{P}_{\text{effector}} (M_{\text{effector} \rightarrow \text{neck}} M_{\text{neck} \rightarrow \text{spine3}}) M_{\text{spine3} \rightarrow \text{world}}$$

$$M_{\text{effector} \rightarrow \text{joint}} = M_{\text{effector} \rightarrow \text{neck}} M_{\text{neck} \rightarrow \text{spine3}}$$

So now a pitch and rotation will be inserted for each joint in the chain in order to orient the effector at the target. The matrix that results from a pitch and then a rotation is the matrix product $Rz4(p) Rx4(h)$. In the coordinate system of the bones, pitch p is a rotation about Z and heading h is a rotation about X. Continuing to use Spine3 as the example, here it is shown in its place with the other matrices.

$$\vec{P}_{\text{world}} = \vec{P}_{\text{effector}} M_{\text{effector} \rightarrow \text{neck}} M_{\text{neck} \rightarrow \text{spine3}} Rz4(p_{\text{spine3}}) Rx4(h_{\text{spine3}}) M_{\text{spine3} \rightarrow \text{world}}$$

The $Rz4(p_{\text{spine3}}) Rx4(h_{\text{spine3}})$ matrix is going to rotate the rest of the chain so that the effector points at the target.

This same process will happen for each joint in the chain. First the eyes will be pointed at the target or as far as they can rotate. Then if the eyes cannot reach, the neck will be turned so that the eyes reach. If still the eyes do not reach then spine3 will turn the neck and eyes even more. So the question becomes, how can a joint earlier in the chain be turned so that the eyes (the effector) point at the target?

4 The Method of Steepest Descent

This is where the Method of Steepest Decent comes in. A function to be maximized will be defined that will be called the "aim". When aim is 1.0, the eyes are pointing directly at the target. When aim is -1, the eyes are pointing 180 degrees in the wrong direction. The idea is that the direction the effector points in must be the same direction as a vector from the target

point to the base of the effector (\vec{P}_0). This calculation is repeated for every joint in its own space.

\vec{P} = target point in local joint space

$$\vec{P}_0 = (0, 0, 0, 1) M_{\text{effector} \rightarrow \text{joint}} \text{Rz4}(p) \text{Rx4}(h) \quad (\text{joint space of origin of effector}) \quad (4)$$

$$\vec{P}_1 = (0, 1, 0, 1) M_{\text{effector} \rightarrow \text{joint}} \text{Rz4}(p) \text{Rx4}(h) \quad (\text{joint space location of effector's tip}) \quad (5)$$

$$\text{aim} = \frac{(\vec{P} - \vec{P}_0)}{|\vec{P} - \vec{P}_0|} \bullet (\vec{P}_1 - \vec{P}_0) \quad (\text{range is } -1 \dots 1) \quad (6)$$

keeping in mind that $(\vec{P}_1 - \vec{P}_0)$ is already a unit vector

The vector $(0, 0, 0, 1)$ in equation (4) is the origin of the effector in its own space. The vector $(0, 1, 0, 1)$ in equation (5) is the direction the effector is pointing in the effector's own space. This vector is one of the quantities that is bound to the coordinate system of the sample animation. If X was forward in the animation then the vector would be $(1, 0, 0, 1)$ instead.

The aim function in equation (6) can be computed using the standard functions in the XNA math library but in the associated XNA project the result was cut/pasted from Mathematica expanded out.

Next we consider what happens to the aim if the pitch p and the heading h are changed. Note that aim is a function of pitch (p) and heading (h). The gradient will be computed as

$$\nabla = \left(\frac{\partial \text{aim}}{\partial p}, \frac{\partial \text{aim}}{\partial h} \right)$$

This is where it was simple to do but very difficult to explain. Mathematica was used for the math so once the expression for the aim was known, it was just a matter of asking Mathematica for the gradient. $D[f[x], x]$ will compute the symbolic derivative of function $f[x]$ with respect to x . So the statement was simply

$$\text{grad} = \{D[\text{aim}, p], D[\text{aim}, h]\}$$

Then one asks Mathematica to replace cosines and sines with cp , ch , sh , cp , etc and then put it into a form suitable for C code called "CForm". The gradient expressions for pitch and heading were then cut/pasted into the C# code. The result had to be modified by hand to not use the C lib function $\text{pow}(x,y)$ which computes x^y but that was trivial.

Mathematica and other Computer Algebra Systems (CAS) are good at simplifying expressions but they are poor at showing the steps useful for explaining. Axiom, and Maxima were investigated to help the reader without Mathematica find a free substitute. wxMaxima was very close to Mathematica, and even the syntax was similar. It installed without any trouble on a Windows XP machine, which is sometimes a problem when free software is involved. If an initiated reader wanted to replicate this work using free software, wxMaxima is recommended.

The gradient unit vector indicates the direction of maximum increase of the aim function. The variables p and h should then be moved as a vector in that direction for quickly maximizing aim. The problem then is how far to move them in that direction. A constant rate was hard coded into the code at what the author thought was an acceptable speed for each joint. Obviously the eyes can move faster than the neck and the constants reflect that. The movement of pitch and heading also was done in a way that ensures the change produced a positive change in the aim. If they are moved too far and the change overshoot the maximum, then the change will actually lower the aim. Overshoot was detected and when it occurred, the step size was divided by two and tried again, with up to four tries.

Mathematica showed that the aim function was smooth for many test cases and was not a high order function that would produce useless local maxima.

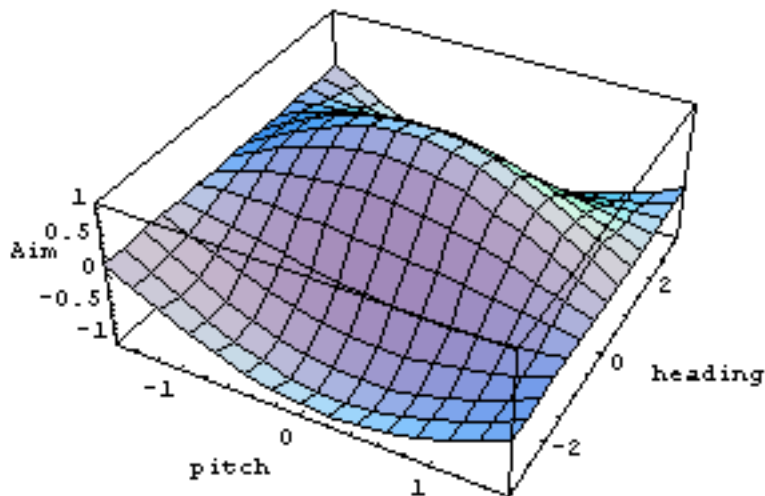


Figure 2. Sample Aim function rendered by Mathematica.

5 Minimizing Joint Displacement (aka The Hack)

The author was very pleased with the results of this experiment. The solution met all the criteria in Section 1. But then it was noticed that the solution does things that people do not. The target point can be moved so that the eyes need the help of the neck to reach the target. Suppose the eyes look all the way left and the neck is turned slightly left. Now the target point is moved to the right. The eyes can follow the path of the target point without the help of the neck for a long way. This means the neck can be pointing left while the eyes are pointing right even though the target point is just slightly right. This solution has no criteria for minimizing the displacement of the joints while pointing the effector at the target.

One possible way to fix that is to add terms to the gradient function to subtract aim by some scaled factor of the joint pitch and joint heading. Essentially, penalize the aim by an amount proportional to the pitch and heading. This was not attempted because it was thought that the penalty would create steady state error in the aim function.

Another way was actually used. When the aim of the eyes is nearly one, then the goal has been reached. Now there is a little more freedom for the supporting joints. In that case, the pitch and heading of each supporting joint was multiplied by 95%. The thinking was that the eyes are already at their target so the supporting joint can attempt to reduce its joint displacement. If this actually reduces the aim then the next frame will correct the aim. This worked well enough to keep.

6 Solution Analysis and Further Work

A frustrating aspect of making professional games with work based on papers is that the papers tend to highlight their benefits and not mention their faults. The authors are trying to get PhDs and research money. When writing as a hobby, one is less motivated to be a salesman. So in that light, here are the pros and cons.

6.1 Pros

Again the visual results are good. Users who compile the sample project should be pleased. The neck and spine3 only assist when they are needed which matches what a person does. Here are the keys.

Moving the targetPoint - I,J,K,L,M,N

Moving the camera - usual keys from the Skinning Sample - Arrow Keys, Z,X

Showing the reference frames of the bones - Q,A

Switch the Effector between the eyes and the neck - E

6.2 Cons

The character does not know how to look the long way around. This means that if the target point orbits the character, his head will be all one way for too long. This can be fixed by clipping the target point to a plane in front of the character. The author did that while working on a popular football game so it is possible. It was not repeated here.

“The Hack” is also a strike against this solution. It would be better if the overall formulation minimized the joint pitch and joint heading. But that solution may lead to much more computation including an overly complex technique from the field of Optimization.

The math here is dependent on the coordinate systems in XNA and the particular animation. To get around this, you can either download and try Maxima to repeat the math, break out pencil and paper and do the derivatives yourself, or try emailing the author for help with a description of your coordinate system. If you happen to work at EA Tiburon, there is a Mathematica license floating around that was purchased for the NASCAR PC team. See the IT staff.

There is only one sample animation to play with in the skinning sample so the testing was not very rigorous.

7 Further Work

Further work of course involves work on the “Cons”. A newer fomulation that considers all the joints at the same time across the entire chain could work better. A better explanation besides “Use Mathematica” is in order. Some work was done on that already but it was an entirely different formulation and then it would require rewriting the sample. The character needs a good way to follow a target point that orbits his head. If people actually read this, the author will find it worthwhile to do this work. His wife though is sick of seeing the thug on his screen and she would rather that he watch 24 with her.

8 Acknowledgements

Thanks to Doug Hayes for saying the key point that the joints needs to all work together to point the effector at the target. It seems obvious now but it didnt' then. He probably doesn't remember saying that.

Thanks to XNA for making such a simple framework for exploring technology in games.

Thanks to ziggyware for hosting my hobby work.

Thanks to my friends that used to work with me on games for talking me into writing this up. Hopefully, someone will find it useful.

Please see this article for more about the matrix notation used here and in the code.
http://www.gamasutra.com/features/20020510/johnson_01.htm

9 Appendix : Mathematica Statements

```

norm[v_] := With[{x = v[[1]], y = v[[2]], z = v[[3]]}, Sqrt[x*x + y*y + z*z]]

Rz4[p_] := With[ { c=Cos[p], s = Sin[p] }, {{c,s,0,0},{-s,c,0,0},{0,0,1,0},{0,0,0,1}}];
Rx4[h_] := With[ { c=Cos[h], s = Sin[h] }, {{1,0,0,0},{0,c,s,0},{0,-s,c,0},{0,0,0,1}}];

effectorToJoint = {{m00, m01, m02, 0}, {m10, m11, m12, 0}, {m20, m21, m22, 0}, {tx, ty, tz, 1}};

targetPoint = {px, py, pz};

p0 = {0, 0, 0, 1} . effectorToJoint . Rz4[p] . Rx4[h]

p1 = {0, 1, 0, 1} . effectorToJoint . Rz4[p] . Rx4[h]

aim = Simplify[Expand[Dot[ targetPoint - p0, p1 - p0]/norm[targetPoint - p0]]]

CForm[aim] /. {Cos[p] -> cp, Sin[p] -> sp, Cos[h]->ch, Sin[h]->sh}

grad = {D[aim, p], D[aim, h]}

gradPitch = CForm[grad[[1]] /. {Cos[p] -> cp, Sin[p]->sp, Cos[h]->ch, Sin[h]->sh }

gradHeading = CForm[grad[[2]] /. {Cos[p] -> cp, Sin[p]->sp, Cos[h]->ch, Sin[h]->sh }

CForm[gradPitch] /. {Cos[p] -> cp, Sin[p] -> sp, Cos[h]->ch, Sin[h]->sh}

CForm[gradHeading] /. {Cos[p] -> cp, Sin[p] -> sp, Cos[h]->ch, Sin[h]->sh}

```