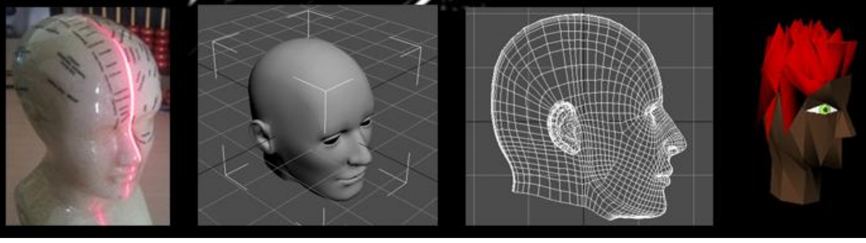
3D Modelling and Animation Research Project

Alex Hughes

2012



3D Modelling has been with us for some years. It has become a part of many industries. From entertainment and computer games to manufacturing, education and medicine. In this research project, I look at the application of 3D technology within the sphere of Education and generally discuss the technical requirements for 3D motion capture, model making and animation.

Chichester College
HND Computing Year 2
Unit 38 3D Modelling and Animation

Uses of 3D modelling and animation within education

The internet revolutionized education. For the first time people were no longer confined to what the tutor knew off the top of their head or what the school-issued textbook had to offer. Hypertext documents could answer any question.

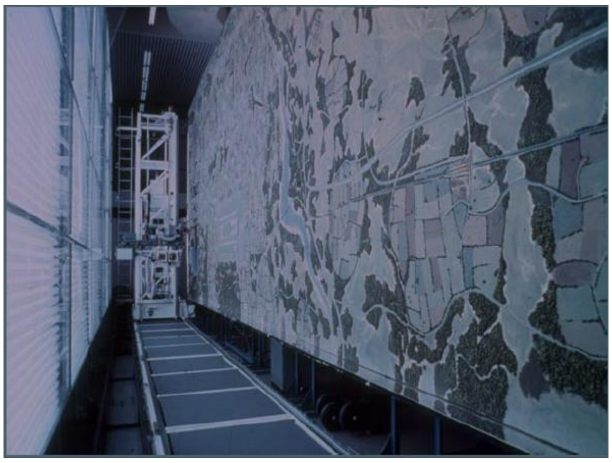
3D modelling could have the same effect.

Early steps

Educational 3D modelling undoubtedly started in Utah with the work of Ed Catmull (who went on to found Pixar Studios) and if you haven't seen it, you really should see this short film which would have taken the University's most powerful computer an age to render.

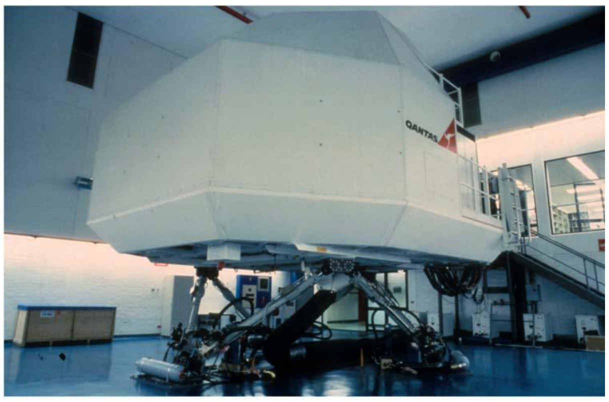
http://vimeo.com/16292363#

Flight Simulators



With literally sky-high budgets, it was the Military who pioneered flight simulator technology. The demand was there even before the technology had evolved with "model boards" being used (where a small video camera is flown over a physical model and the resultant video presented to the trainee pilot.

The first full visual systems were produced by the General Electric Company for the space program and featured patterned ground objects. However it was well into the 1970's before the model-boards were retired and CGI flight simulators became common place.

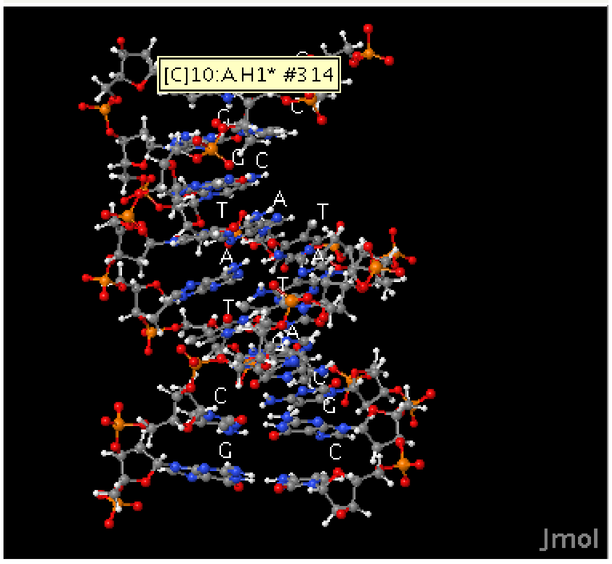


Modern simulators use a 180 degree curved projection screen and hydraulics to simulate acceleration.

Despite the great expense airlines and the Military have always been keen to invest in Flight Simulation. Safety and expertise are paramount in these industries and scenarios can be simulated that would never even occur in training sessions.

(Page) Breif History of Flight Simulation

Molecular Models and Biology



When the structure of DNA was discovered in 1953 by Rosalind Franklin, Francis Crick and James Watson, it could only be visualized with a massive stick and ball model.

Nowadays, the standard visualizer for chemistry hyper-documents is jMol. This java applet runs on nearly any platform and takes a variety of formats for chemical formulas.

This allows users not only to see a model but also to rotate and enlarge it in real time.

More sophisticated molecular modelling programs exist in the field of drug design and biochemistry.

It is still impossible to understand how enzymes (the human chemicals responsible for actually performing the tasks of a living cell) interact with chemicals. However using molecular models, the reactions can be simulated.

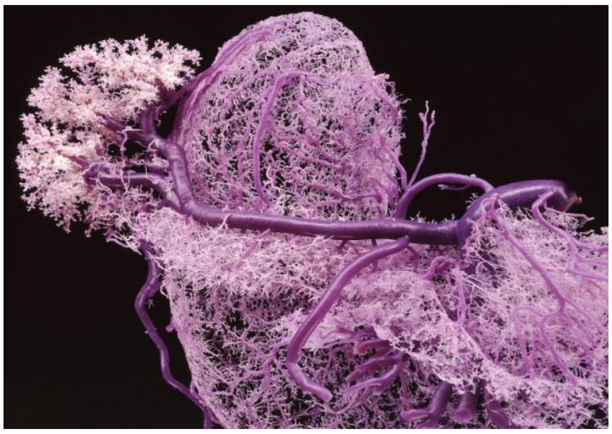
Applications for this include Amber from University of San Francisco.

This molecular dynamics program takes Cartesian coordinates of each atom in a molecule and Topology information (i.e. which atoms are bonded together) and Force fields (which is information about the exact electromagnetic and other fundamental forces generated by the bonds and atoms in the molecule) and uses it to simulate reactions.

Combined with a graphical interface this allows students to see how things happens and is an invaluable tool for designers of bio-molecular processes to create new chemicals which may have potential as drugs.

http://ambermd.org/doc12/AmberTools12.pdf (UCSF)

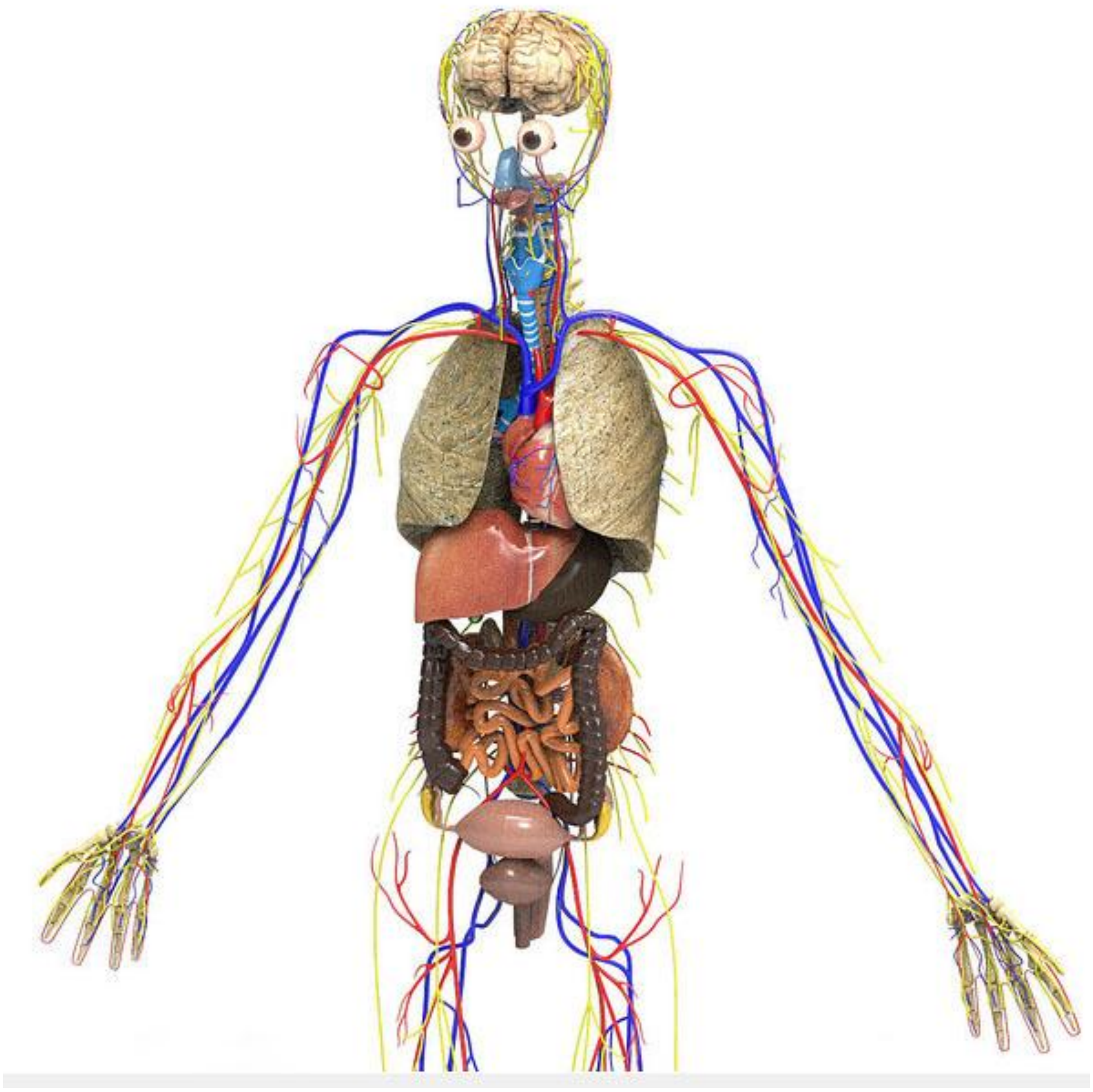
When I was at school, models of the human vascular system used to be created literally by filling the



veins of a research body with latex and then dissolving off the flesh in acid. There was no other way of creating them.

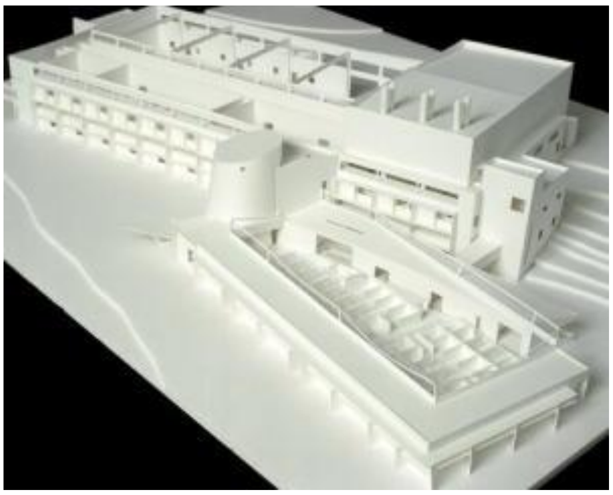
Now 3D models for all anatomical views have been created and are an invaluable tool for biology classrooms.

Another advantage of the models is they allow students to learn about anatomy at an earlier age as dissection becomes only necessary for much more advanced studies.



http://www.turbosquid.com/3d-models/3d-female-circulatory-heart/694548(Polygon Puppet)

Architecture



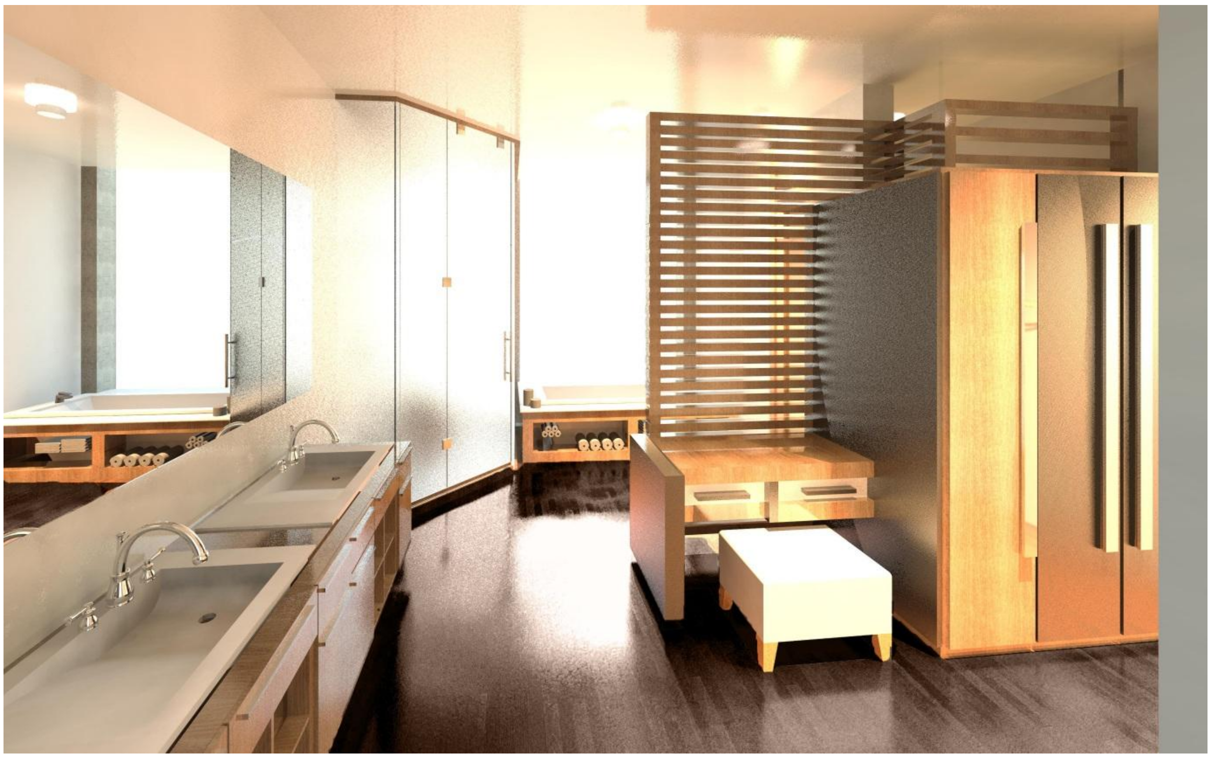
Of all the people who study to become architects, only a small percentage go on to create their own visions as public buildings. It is a 6 year degree course and a great deal of the course used to comprise making models out of paper and card.

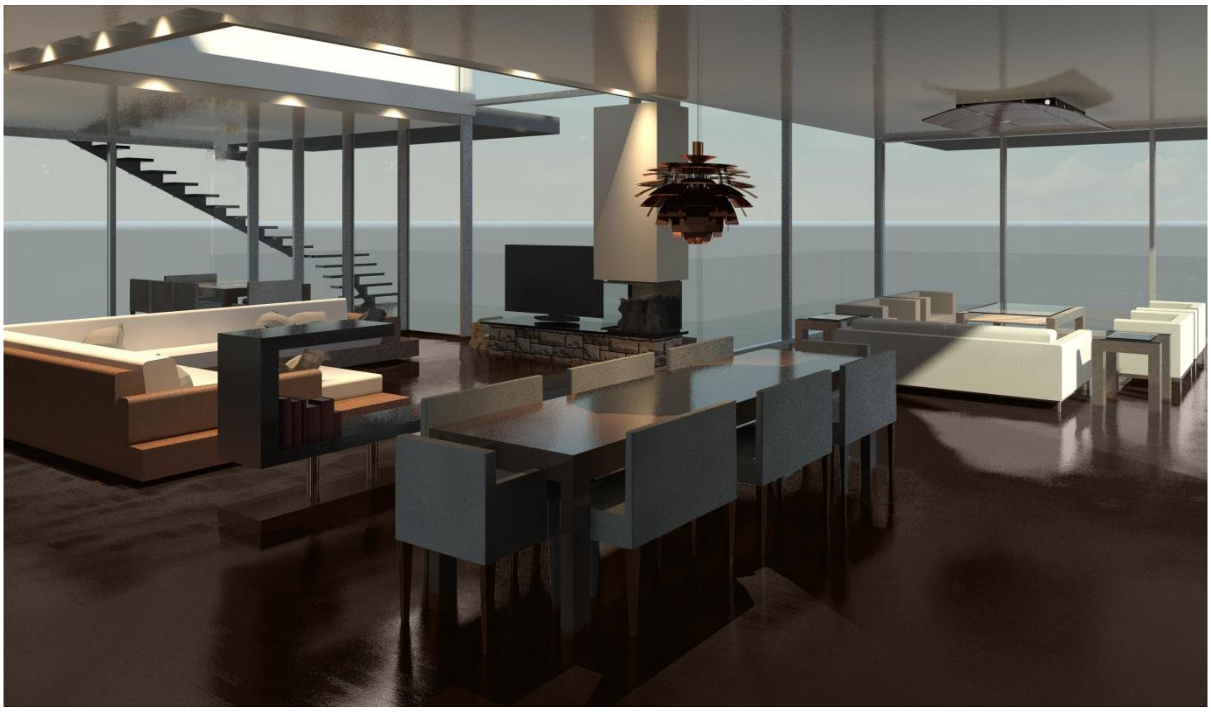
This used to be time consuming and not all that realistic with regard to materials.

3D modelling has revolutionized the field.

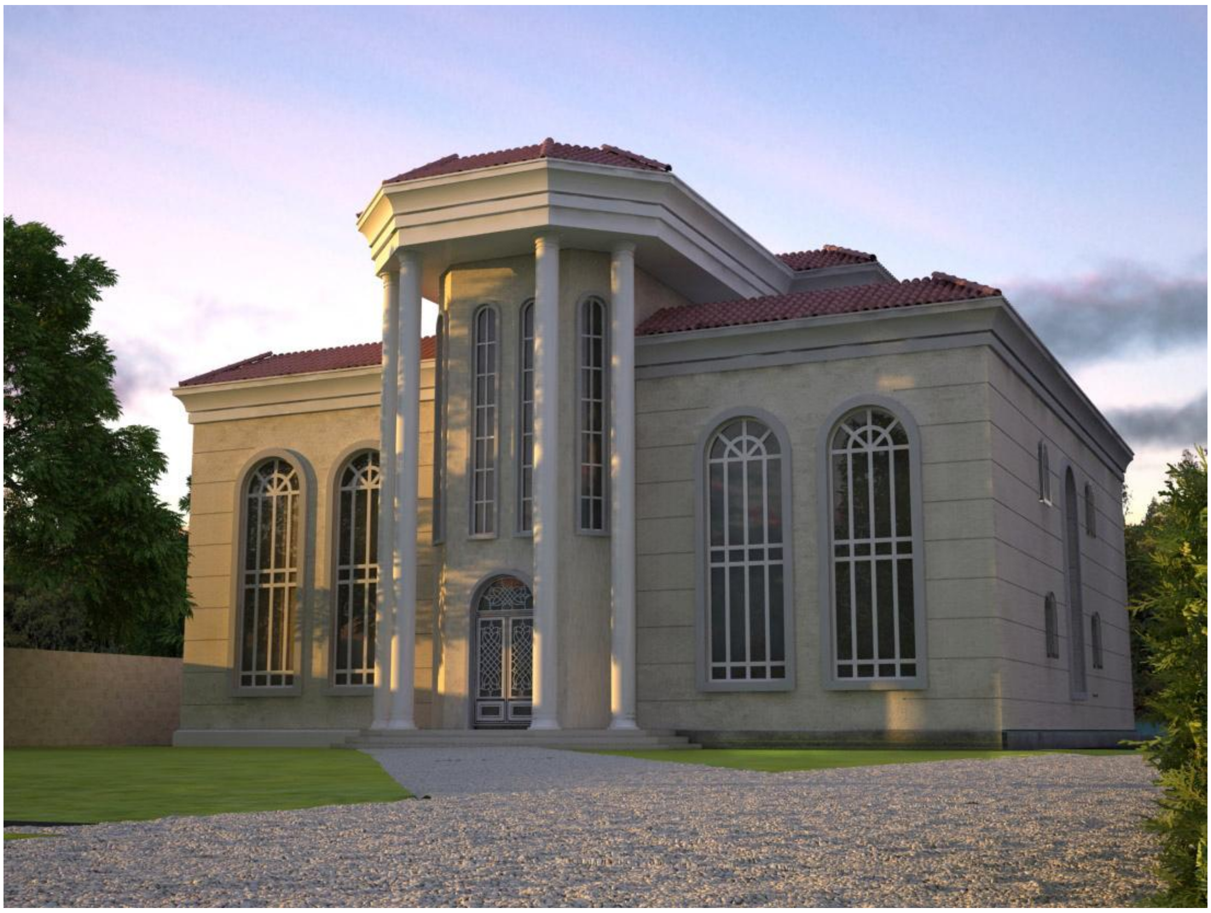
Autodesk, the company who originally created the CAD/CAM package AutoCAD, have diversified the product almost unrecognisably from its engineering origins.

AutoCAD provides support for a vast library of materials available to the building trade from glass bricks to steel girders. Architectural designs can be drawn up and visualized as they are being created exactly as they would appear when complete.





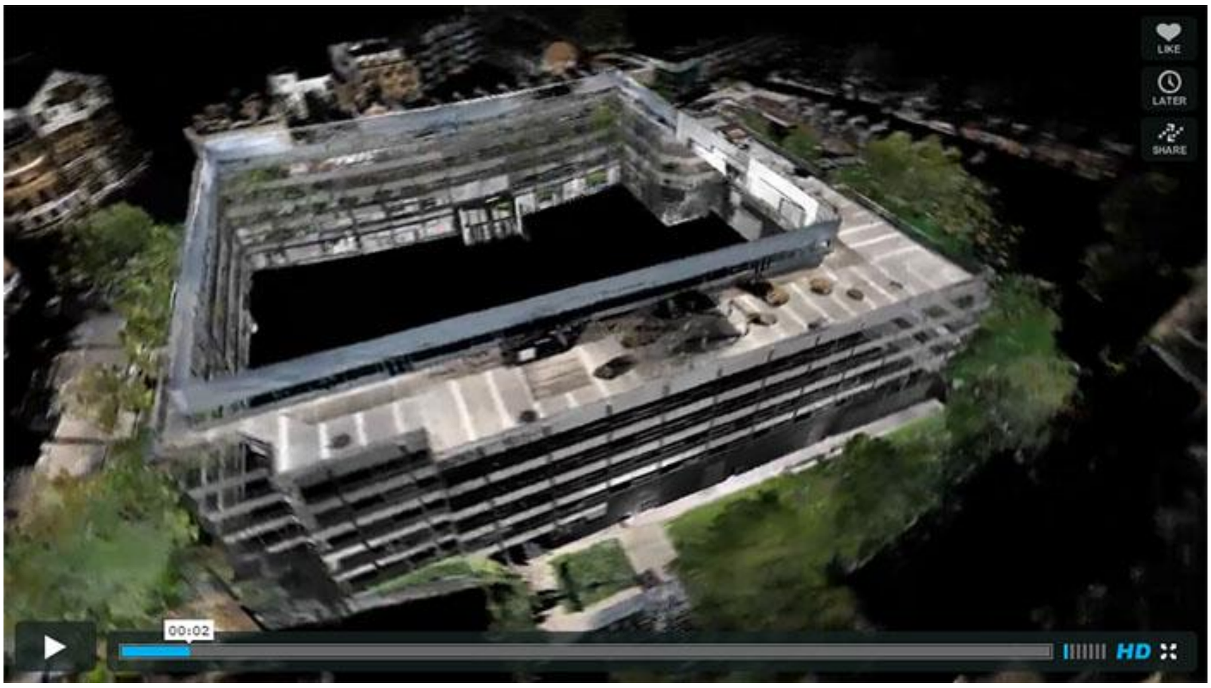
student work(Moderbacher)



student work (Sabry)

ScanLab

Based at UCL's Bartlett School of Architecture, ScanLAB is a project dedicated to producing 3d models through laser scanning.



http://vimeo.com/30737412(ScanLAB)

The models are made by a combined photographic and laser imaging. A colour photograph is taken of the scene. Then, through the same lens a video is taken of a laser scanning across the scene from a different angle. Because the origin and direction of the laser beam is known, the point at which it hits an object will be displaced by parallax so its 3d coordinates can be calculated (more on this later). When you know where a voxel is and you know its colour you can add it to your scan.

It takes many scans to build a model and the work is usually done at night to ensure a clear image of the laser spot. Once the scans are done, the hard work of stitching the scans together begins.

The Techniques

Model Making

Nowadays there are many different tools for creating 3D models. From Autodesk Max and Maya to Daz3D and Poser. Of particular interest to anyone who wants to do this commercially without much outlay is Blender – an Open Source 3d graphics environment project based in Holland. The Blender Foundation also works with up and coming film-makers and you can submit ideas to a competition. If you win you can get an expenses paid trip to Amsterdam to develop your project and work on putting together a team from members of the community. See http://projects.blender.org/.

The techniques for 3d sculpting are many and diverse and too numerous to be mentioned here. There are several websites devoted to selling 3d models for games films and just posing scenes.

One advance I look forward to is virtual reality modelling using Vuzix Augmented Reality glasses.

The Wrap 920AR glasses present Virtual Reality in 920 line resolution and at the same time capture video through stereoscopic cameras.

This makes 6 degrees of freedom head tracking accurate and easy.

What is more you could use the camera's output to capture the position of a fingertip or tool in 3d (especially with other cameras used as well)

This could provide for literal 3D sculpting.

The maxReality interface is already written (though expensive at £1500 for glasses plus software).

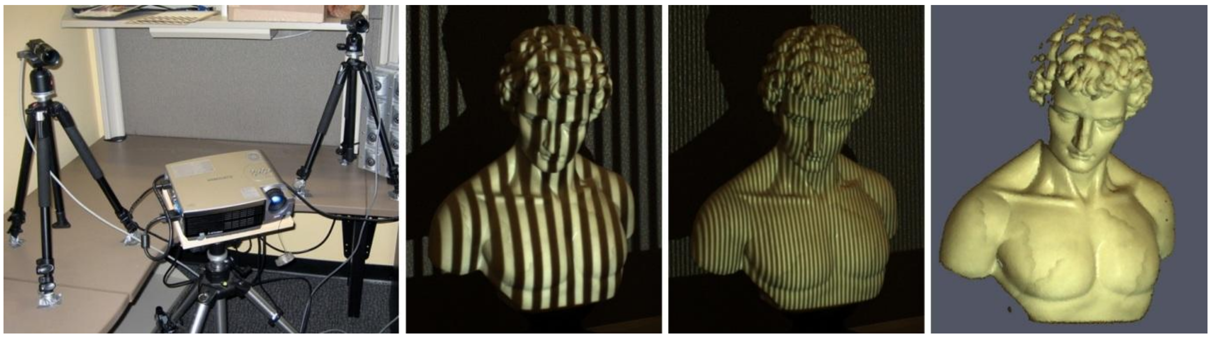
Cameraless VR wrap glasses come in at £399.99 with an attachable single usb camera at £39.98 which seems like it is calling out for someone to hack blender to use this.

3D Scanning

Several devices exist that will do 3D scanning.

Structured Light

Structured light scanning is done with a video projector and digital cameras.



http://mesh.brown.edu/3DPGP-2009/homework/hw2/hw2.html (Brown University)

The projector emits a series of patterns of stripes starting with all white, then half white / half black, then quarters and so on.

The cameras (at offset angles) pick up the boundary between light and dark stripes and get the 3d shape by parallax (wikipedia).

This technique works best on static objects because you have to get a lot of overlaid images. For 1024 resolution you will need 10 images at 24 fps capture rate that is nearly half a second.

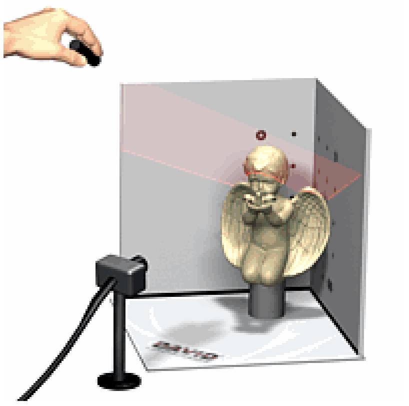
You can do better than this. In fact you can do a lot better. Video Projectors work on an element called a Digital Micro Mirror Device. Because the lcd can be only on or off, 256 shades of red, green and blue are produced by flashing the element on and off for varying lengths of time (pulse width modulation PWM (Wikipedia)). In fact, if you only want white or black a DLP projector is capable of generating over 50,000 binary images per second. http://johnnylee.net/projects/thesis/ (Lee)

So your limitation is more likely to be your camera.

As a rule of thumb, if the total movement of the object throughout the process is less than a quarter the width of the smallest graduation, good software should be able to sort it out.

Laser Scanning

In laser scanning a line laser projects a thin line across an image onto a calibrated back screen. The offset camera can then resolve the shape of the 3d model.



You can set up a home scanner using a laser for minimal cost. You need the laser, calibrated background and a camera. Software is available from http://www.david-laserscanner.com/ (David Lasercanner) to process both laser and structured light scans. It is recommended not to scan human beings with a laser.

Commercial laser-scanning devices are still expensive but are suitable for scanning larger areas such as architecture and crime scenes.

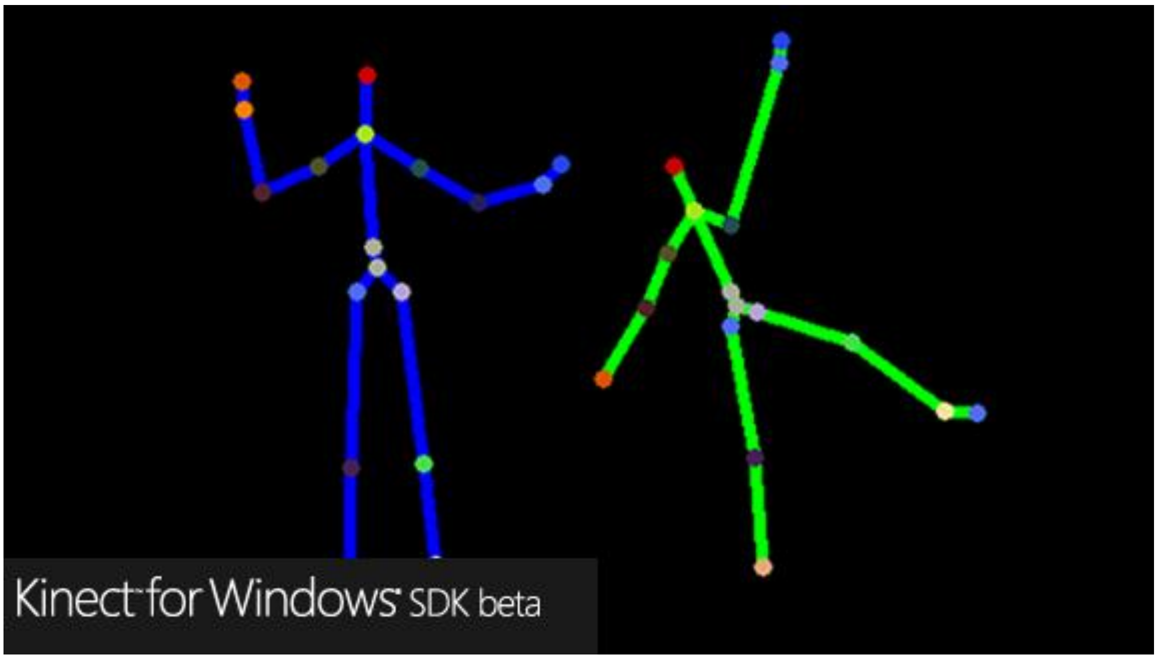
http://www.faro.com/usa/contentpages/colorscanner/

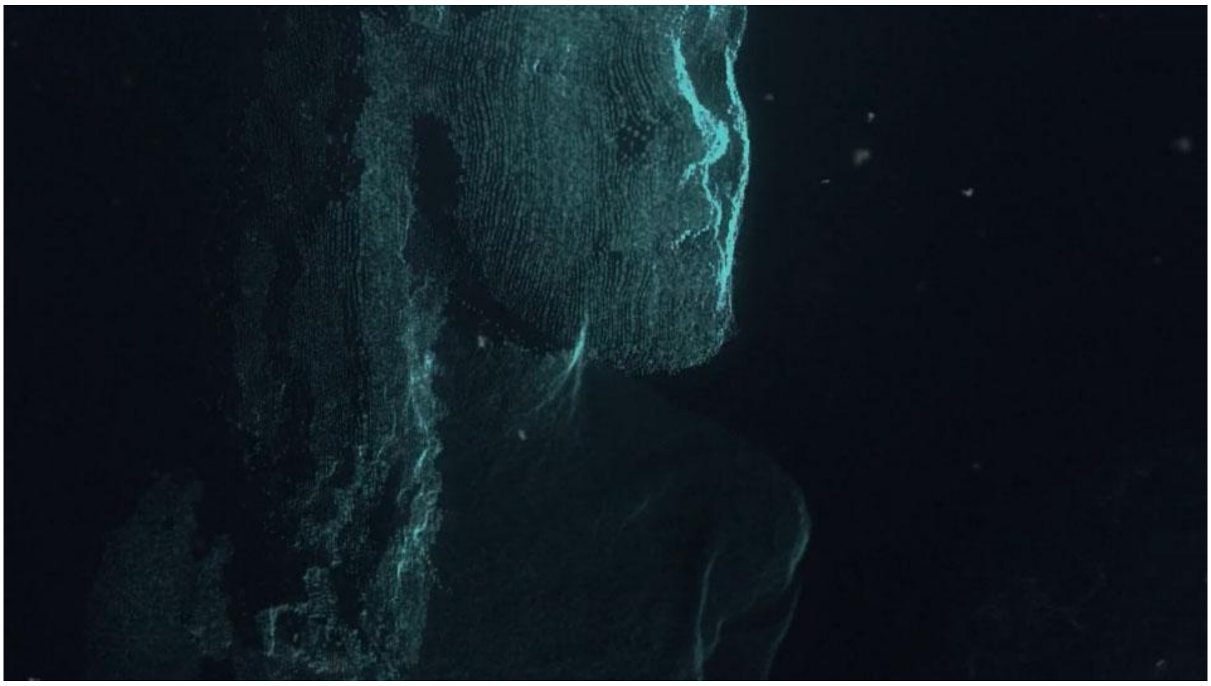
Kinect for Windows

Microsoft's newly revamped, newly re-priced Kinect scanner is actually incredibly useful to the home hobbyist.

Apart from the price, the Windows version differs from the xBox version in having a near range setting, an extra chip and the software to connect to windows.

At nearly £200 pounds, it is not cheap but allows for basic motion capture, 3d scanning and control of models in real time.





New Look - Nap On The Bow (Tim & Joe)

Here is an example of a pop video made using Kinect Camera.

http://vimeo.com/30427082

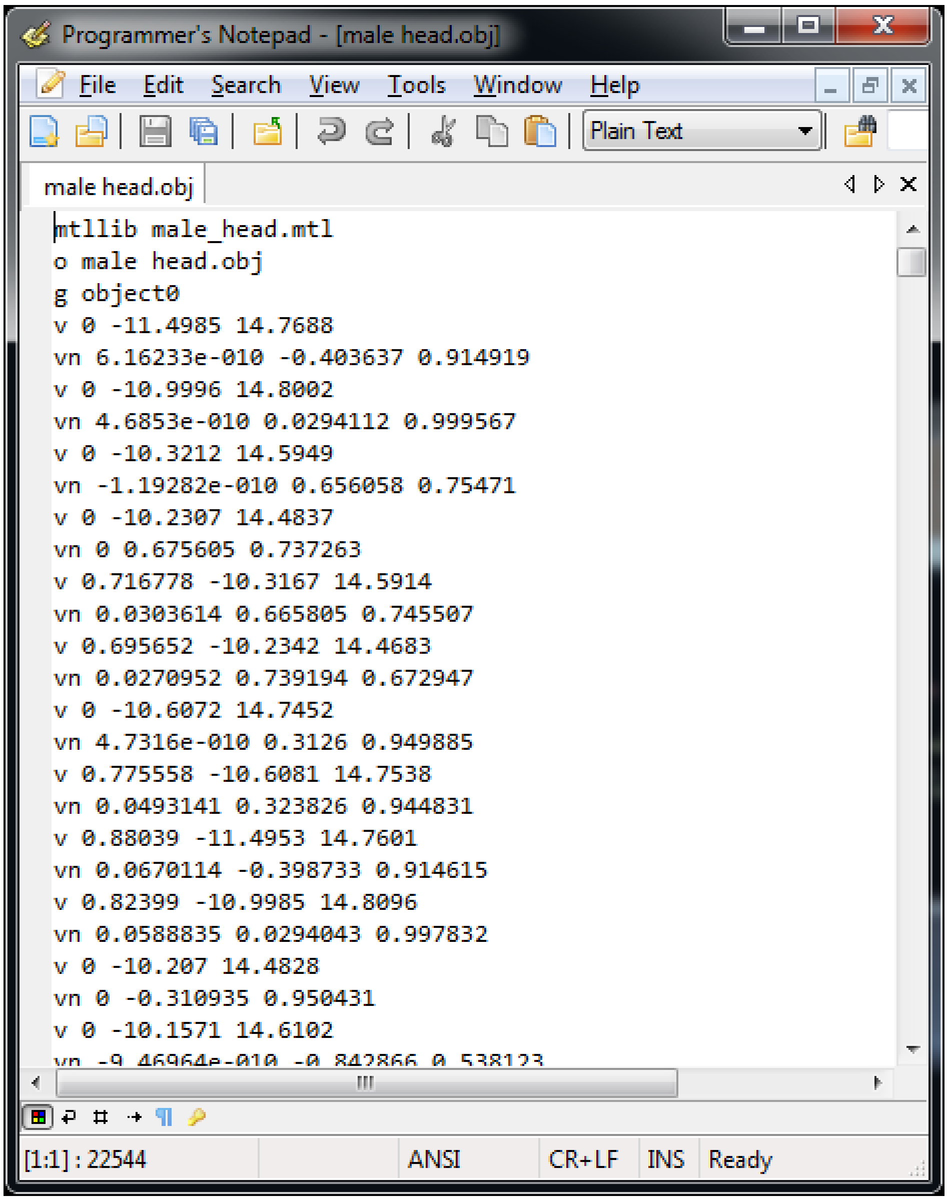
Interactive Puppet Prototype with Xbox Kinect (Watson)

http://vimeo.com/16985224

The Mathematics and Computing of 3D

The .obj file

There are many ways of representing a 3D model but .obj files are by far the most user friendly. The coordinates are presented in ASCII format with groups, materials, vertices, normal all referenced by simple letter descriptions at the beginning of the line.


Programmer's Notepad - [male head.obj]
File Edit Search View Tools Window Help
Plain Text
male head.obj
[1:1] : 22544
ANSI
CR+LF
INS
Ready

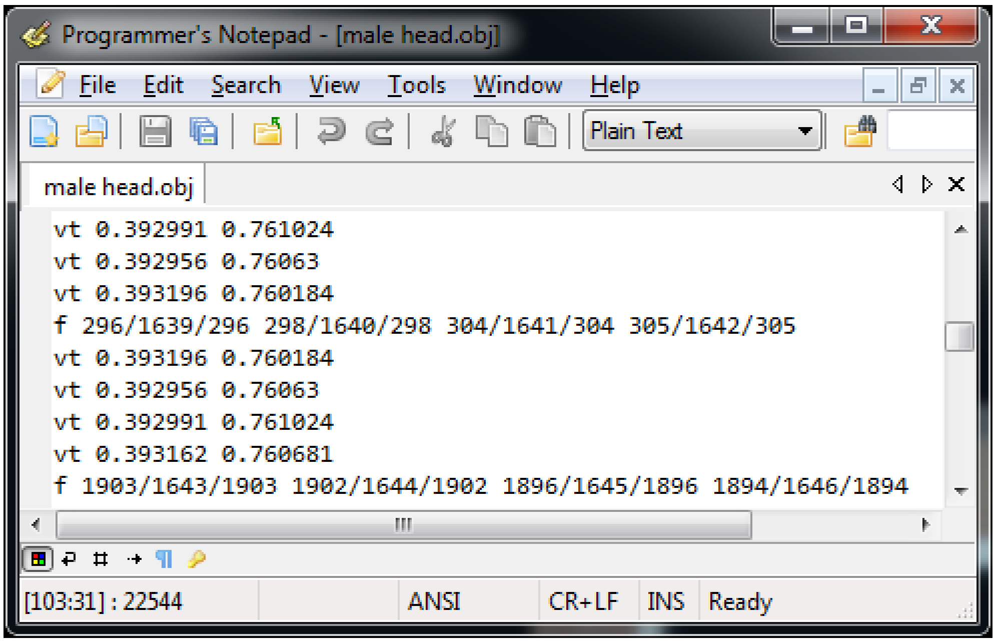
```
mtllib male_head.mtl
o male head.obj
g object0
v 0 -11.4985 14.7688
vn 6.16233e-010 -0.403637 0.914919
v 0 -10.9996 14.8002
vn 4.6853e-010 0.0294112 0.999567
v 0 -10.3212 14.5949
vn -1.19282e-010 0.656058 0.75471
v 0 -10.2307 14.4837
vn 0 0.675605 0.737263
v 0.716778 -10.3167 14.5914
vn 0.0303614 0.665805 0.745507
v 0.695652 -10.2342 14.4683
vn 0.0270952 0.739194 0.672947
v 0 -10.6072 14.7452
vn 4.7316e-010 0.3126 0.949885
v 0.775558 -10.6081 14.7538
vn 0.0493141 0.323826 0.944831
v 0.88039 -11.4953 14.7601
vn 0.0670114 -0.398733 0.914615
v 0.82399 -10.9985 14.8096
vn 0.0588835 0.0294043 0.997832
v 0 -10.207 14.4828
vn 0 -0.310935 0.950431
v 0 -10.1571 14.6102
vn -9.46964e-010 -0.842866 0.538123
```

Explanation of the file.

mtllib refers to a material file. this could be a plain Lambert render or a texture map.

v refers to a vertex in the model

vn refers to the normal of the vertex (this is useful in deciding what colour the polygon will be shaded and is also used when shape smoothing is carried out.


Programmer's Notepad - [male head.obj]
File Edit Search View Tools Window Help
Plain Text
male head.obj

```
vt 0.392991 0.761024
vt 0.392956 0.76063
vt 0.393196 0.760184
f 296/1639/296 298/1640/298 304/1641/304 305/1642/305
vt 0.393196 0.760184
vt 0.392956 0.76063
vt 0.392991 0.761024
vt 0.393162 0.760681
f 1903/1643/1903 1902/1644/1902 1896/1645/1896 1894/1646/1894
```

[103:31] : 22544
ANSI
CR+LF
INS
Ready

Once we get into the faces,

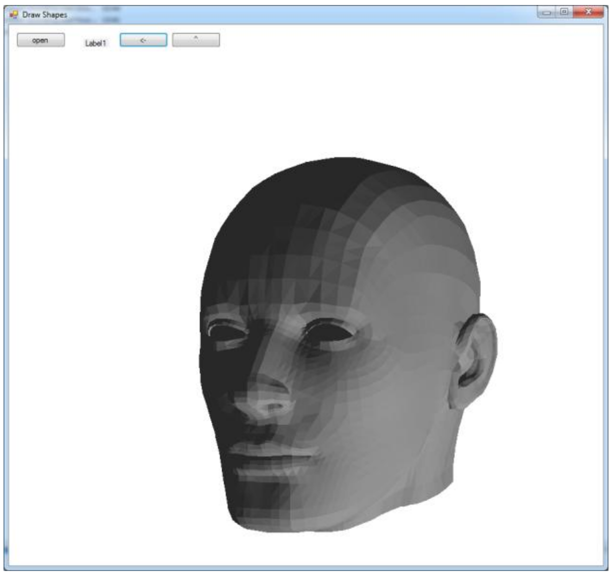
f represents the definition of a new face and the 3 number separated by slashes (/) represent, the vertex number (v) the index of the 2d texture coordinate of that (vt).

and finally, the index of the normal for that vertex (vn).

Here the texture map coordinates have been added at each face but they can come earlier in the document too. It is the sequential number that counts.

For a full specification of the .obj format see http://www.martinreddy.net/gfx/3d/OBJ.spec (Reddy)

I have written a program in Visual Basic for Parsing and Rendering 3D content (quite badly as it goes – I was in the Nat Dip year at the time) but it is relatively easy to import objects into WCF just by parsing and then use the more powerful 3D commands in the language to render. Here is the output of my first shader.



Transformation Matrices

We all know the principal behind hierarchical modelling – "The head-bone's connected to the neck-bone …" and so on. But how is it actually done in real applications.

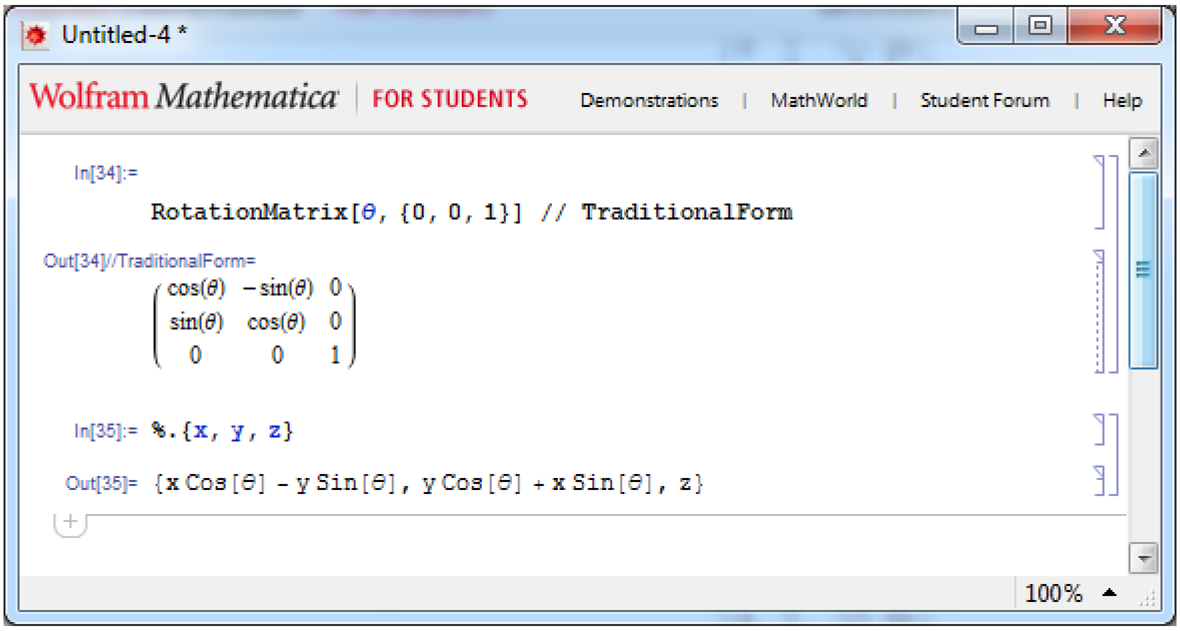
3D points or vertices are represented by vectors in Mathematics which are *tuples* of 3 numbers.

We are provides us with various techniques for manipulating vectors.

Translation: if you wanted to move a point 2cm in the x direction 3cm in the y direction and 4 cm in the z direction you would add those numbers to its coordinates right?

What you have just done is **vector addition**. OK, so we can get our fighter flying through the air whilst standing bolt upright.

Rotation. The traditional way to rotate a vector about a given axis is by a 3x3 **matrix multiplication**. Here is the output for the 3x3 matrix for rotating through angle theta radians about the z axis.



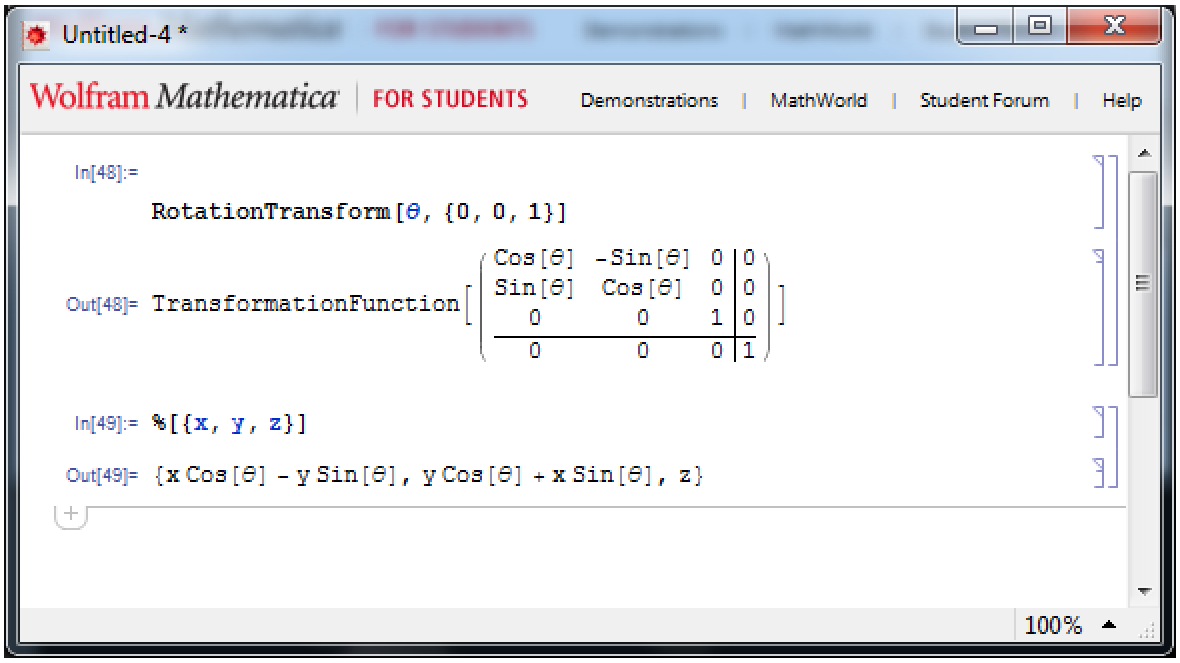
```
In[34]:=
        RotationMatrix[θ, {0, 0, 1}] // TraditionalForm
Out[34]//TraditionalForm=
```

$$\begin{pmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{pmatrix}$$

```
In[35]:= %.{x, y, z}

Out[35]= {x Cos[θ] - y Sin[θ], y Cos[θ] + x Sin[θ], z}
```

The problem was that if you wanted to do a translation and a rotation you had to do separate operations.

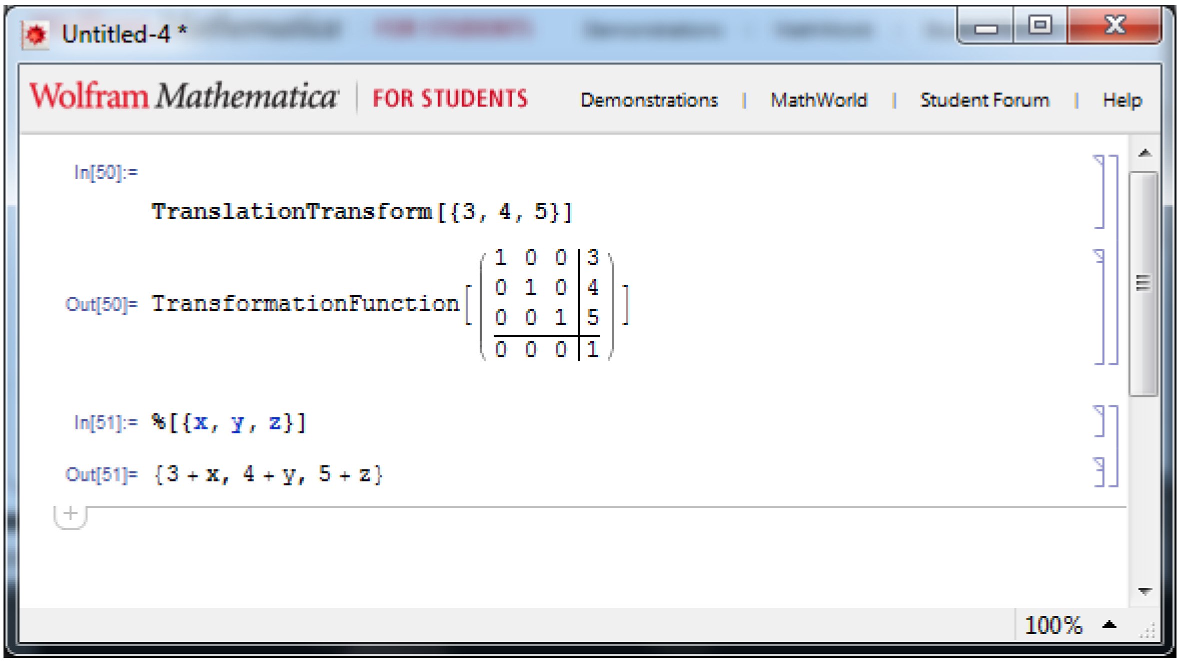
So the 4D transformation Matrix was created.


Untitled-4 *
Wolfram Mathematica FOR STUDENTS
Demonstrations | MathWorld | Student Forum | Help
100%

```
In[48]:=
        RotationTransform[θ, {0, 0, 1}]

Out[48]= TransformationFunction[( Cos[θ]  -Sin[θ]  0 | 0 )]
                                ( Sin[θ]   Cos[θ]  0 | 0 )
                                (   0        0     1 | 0 )
                                ( ---------------------- )
                                (   0        0     0 | 1 )

In[49]:= %[{x, y, z}]

Out[49]= {x Cos[θ] - y Sin[θ], y Cos[θ] + x Sin[θ], z}
```

Here the rotation matrix again just adds an extra column and row with just a 1 in the corner.

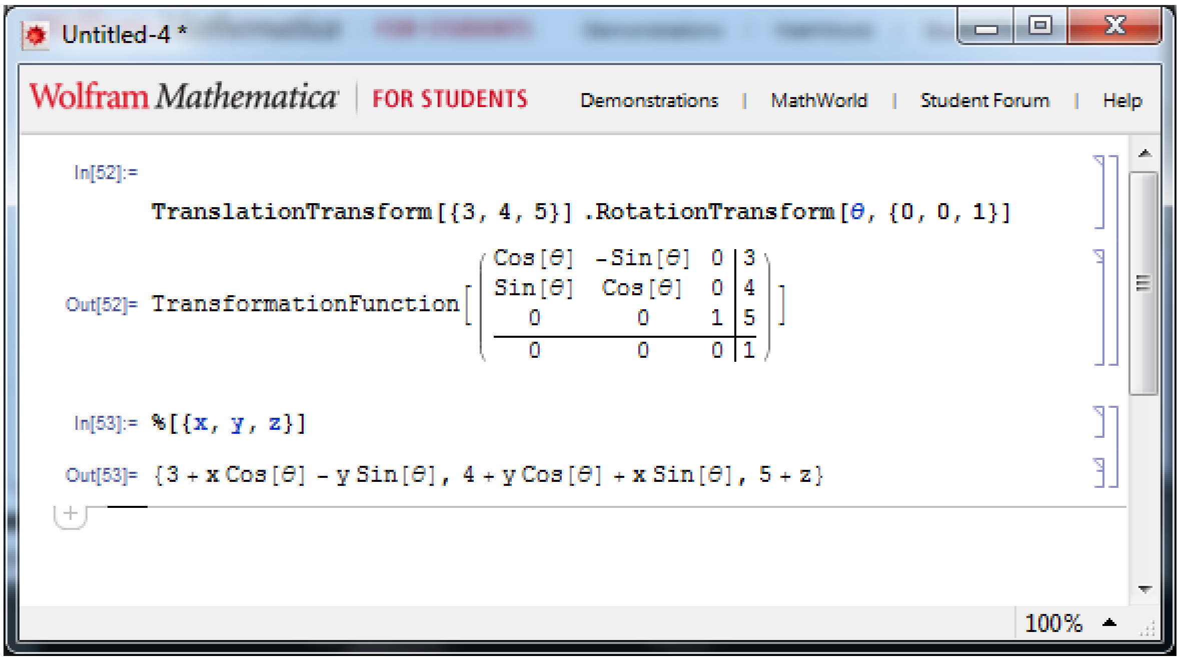
But what is useful is that translation can be done as a multiplication


Untitled-4 *
Wolfram Mathematica FOR STUDENTS
Demonstrations | MathWorld | Student Forum | Help
100%

```
In[50]:=
        TranslationTransform[{3, 4, 5}]

Out[50]= TransformationFunction[( 1  0  0 | 3 )]
                                ( 0  1  0 | 4 )
                                ( 0  0  1 | 5 )
                                ( ----------- )
                                ( 0  0  0 | 1 )

In[51]:= %[{x, y, z}]

Out[51]= {3 + x, 4 + y, 5 + z}
```

The way this works is the 3D vector actually has a fourth coordinate which is always 1 and the translation matrix is actually multiplying various numbers of 1s and adding them on.

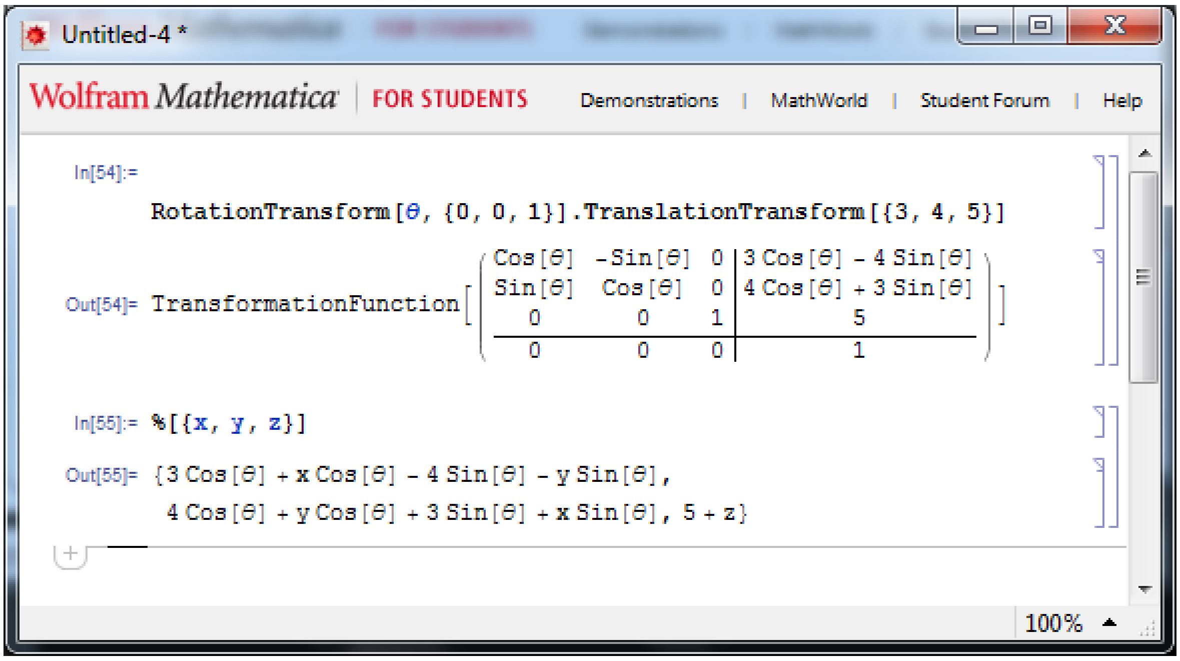
So now the two transforms can be piggy-backed.


Untitled-4 *
Wolfram Mathematica FOR STUDENTS
Demonstrations | MathWorld | Student Forum | Help
100%

```
In[52]:=
        TranslationTransform[{3, 4, 5}].RotationTransform[θ, {0, 0, 1}]

Out[52]= TransformationFunction[( Cos[θ]  -Sin[θ]  0 | 3 )]
                                ( Sin[θ]   Cos[θ]  0 | 4 )
                                (   0        0     1 | 5 )
                                ( ---------------------- )
                                (   0        0     0 | 1 )

In[53]:= %[{x, y, z}]

Out[53]= {3 + x Cos[θ] - y Sin[θ], 4 + y Cos[θ] + x Sin[θ], 5 + z}
```

The last operation to be performed on the vector is written first.

Notice how it is not the same to rotate about the z-axis and then translate through {3,4,5} as it is to translate through {3,4,5} and then rotate.


Untitled-4 *
Wolfram Mathematica FOR STUDENTS
Demonstrations | MathWorld | Student Forum | Help
100%

```
In[54]:=
        RotationTransform[θ, {0, 0, 1}].TranslationTransform[{3, 4, 5}]

Out[54]= TransformationFunction[( Cos[θ]  -Sin[θ]  0 | 3 Cos[θ] - 4 Sin[θ] )]
                                ( Sin[θ]   Cos[θ]  0 | 4 Cos[θ] + 3 Sin[θ] )
                                (   0        0     1 |         5           )
                                ( ---------------------------------------- )
                                (   0        0     0 |         1           )

In[55]:= %[{x, y, z}]

Out[55]= {3 Cos[θ] + x Cos[θ] - 4 Sin[θ] - y Sin[θ],
          4 Cos[θ] + y Cos[θ] + 3 Sin[θ] + x Sin[θ], 5 + z}
```

Once you have the matrix right for the upper arm (which in general will be 3 rotations and a translation) you can piggy back the lower arm off of it.

Here is a good discussion of Matrix Transformations for 3d programming.

http://www.codeproject.com/Articles/42086/Space-and-Matrix-Transformations-Building-a-3D-Eng

http://msdn.microsoft.com/en-us/library/ms753347.aspx

Rendering Techniques

Once the mesh has been manipulated into the shape you want, it is time to render.

The first thing to be applied is 1 more matrix transformation for the position of the camera. This will be a translation and a rotation and sets the camera at a zero position and the scene around it.

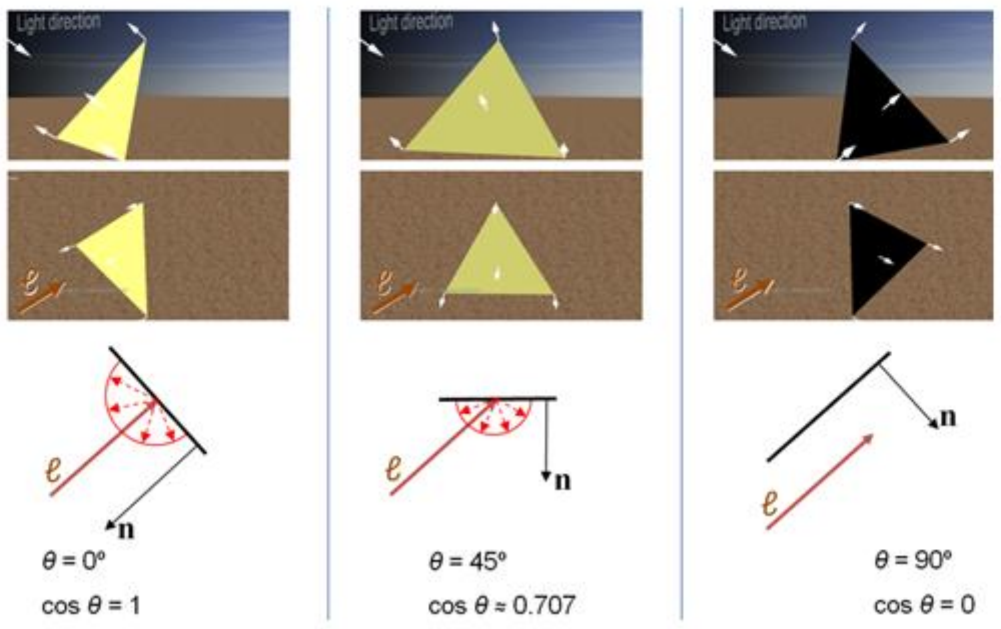
Then a filter is applied so that only polygons within the field of view will be rendered and then the scene is put through a projection matrix. This maps the 3d coordinates to a 2d plane.

The winding order of polygons is important (to save time rendering the inside of solid objects (which should never be seen) polygons are rendered on the “heads” side only). To determine which side is which you look at the winding order. If you look at each point in the order in which it appears in the polygon on your 2d screen. If they go anti-clockwise it is an anti-clockwise winding order and the polygon is rendered. Otherwise the system will not bother.

Next comes the z-ordering. Obviously polygons which are behind others will be hidden and polygons which are partially obscured must be part rendered. Where many renderers go wrong is when polygons actually cut each other and this sort of model should be avoided.

The actual colour to render a polygon can be defined most simply by a law called Lamberts law of Reflection.

This states that the intensity of light reflected from an object is proportional to the cosine of the angle from the normal to the surface and the incident beam.



For a detailed discussion on rendering see
http://www.cs.brown.edu/courses/cs123/lectures/03_GL_3D.pptx (Brown University)

The code in VB to my renderer is included as an appendix. I’m not sure if I even speak VB anymore and will try to write a more logical one later in the year if time permits.

Virtua Project

I have for some time been working on an implementation of Virtua Fighter 1 in order to use these Matrix techniques on a hierarchical model. It is a bit retro as now bone-based animation has taken over but I thought it would be a good learning resource and I had hoped to have a slider bar mock up of the character done for this assignment.

The model is ultra low poly so lends itself to experimentation in WPF.

Unfortunately time has run out and all I have are the head and arms so I will simply include this teaser.

Motion Capture

Visual Tracker Motion Capture

We were lucky enough to go on a departmental visit to Portsmouth University's Motion Capture Suite where Alex Coulson demonstrated the techniques for motion capture using visual markers.

The suite comprises 12 cameras which work in near infrared synchronised with LED flash at 120Hz. Each camera costs around £5000.

The cameras capture high reflectivity markers (polystyrene balls wrapped in scotch hi visibility tape). Even the markers are expensive in mocap. The cameras are 1MP and can resolve the scene to sub millimetre accuracy. They do this by circle centring. The camera outputs a circle where the ball is and the software takes the centre of that circle.

The system works by seeing each marker on two or more cameras at the same time and then **solving** its position in 3d space by parallax. The difficulty lies with multiple markers and knowing which is which between the different images on the cameras. As markers become obscured on 1 camera and appear on the next the computer must keep track of them.

Another difficulty comes from spherical imperfections in the lens of the camera and camera position movement.

Calibrating the system

If the positions of the cameras were set in stone and they behaved like perfect pin-hole cameras solving would be easy. However slight movements and the spherical nature of optical systems leads to barrel and pincushion distortion in the image. see http://en.wikipedia.org/wiki/Distortion_(optics) (Wikipedia)

The system can work with this. It does this by a calibration process.

The first thing the operator does is to take a "wand" – a fixed metal object with 4 balls on it – and wave it throughout the space to be captured at various angles and positions. This allows the system to gauge what standard distances and angles look like and build a correction algorithm.

The wand is then placed on the floor to set the origin and ground level (there is nothing more embarrassing than a mocap in which the character's feet do not touch the ground.)

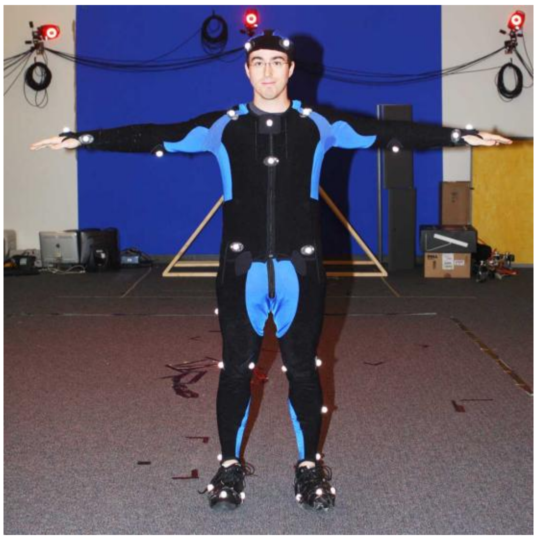
Attaching the markers

because the cameras must have a clear view of the markers, actors wear lycra suits. The markers are attached individually at the start of the shoot. It takes 53 markers to capture a human body (without fingers or face).

Because it would be impossible to attach a marker at the centre of a joint, they are placed on the axis of rotation of joint at either side, it is the job of the computer to figure out where the centre of the joint would lie in 3d space.

Once the actor is marked up, it is time to calibrate again.

Calibrating for the actor



The actor first adopts the so-called T-Pose. This allows the system to measure the lengths of their limbs and body dimensions. They then perform a series of movements to calibrate the system for the position of the markers on their bodies.

Capturing actual motion

Once the system has been calibrated for the actor they can move around and the markers on their bodies will appear on the screen (usually as white dots) but the computer is able to determine which dot is attached to which part of the actor's body (a technique known as **tracking**)

Then the computer can interpolate the position of each joint in the skeleton in real-time. This is known as **solving.**

If 1 marker gets obscured it is usually not a problem, if too many get hidden the model eventually breaks down and an attempt at solving is displayed which usually looks more like an artefact.

If something drastic happens like a marker falling off during capture the scene may need to be reshot. Often however they allow the scene to continue in the hope that something may be captured as you can always "get the trackers to sort it" The trackers are people who work in post-production cleaning up mocap data.

To save disk space, the cameras do not record in full motion video. Rather they only record in monochrome and as the signal from the marker is super-bright, a threshold level is applied and the recorded video will be a pure black and white image of just the circles for the markers.

Live Rendering

Although most motion capture data is cleaned up then applied to a model later on, it is possible to animate and render a model live using Autodesk Motion Builder. This allows a real-time display of the action which can even be interpreted by a virtual camera (literally a video screen attached to 4 markers on a metal frame). The suite captures the position of the "camera" and then feeds the appropriate video to the monitor. The cameraman can then get the correct shot.

Motion can be reshot after the actors have left to get the optimum angle.

Motion Capture without Trackers

It is possible, though hard to capture motion without trackers. At Stanford University , the Markerless Motion Capture Project (Stanford University) has done just that.

The project captures video from all angles as before but instead of looking for markers, it creates a 3d object called a **hull**. This is done by taking the 3d cones projected from each camera's lens to the outline of the actor and extending them, the cones are then intersected to produce a hull that will contain (hopefully quite tightly) the actor's form.

The hull is then compared to a known database built up of 3d models of the actor in variable poses and the system works backwards to solve for the skeleton.

The process is highly processor intensive and may have issues around clusters of people but allows actors to wear their own clothes which are captured as well.

Portable Motion Capture with moving sensors

Mocap is not just for film and games.

An interesting structured light approach has been pioneered at MIT and is used to as an educational resource improve Motor skills.

A structured light gray code is projected, not using a DLP projector but by an array of 8 IR diodes. The markers in the suit are actually passive light detectors similar to those found in TV remote controls. They ascertain their position in x and y relative to the projector from the light signals they receive. Using 2 or more projectors mocap can be done in 3D.

The markers then relay their position to the computer via radio. The whole marker circuit is less than the size of a US coin.

There is a no limit to the number of markers you can have because they work in parallel and solving is easier because you already know which marker is which (the tracking is already done).

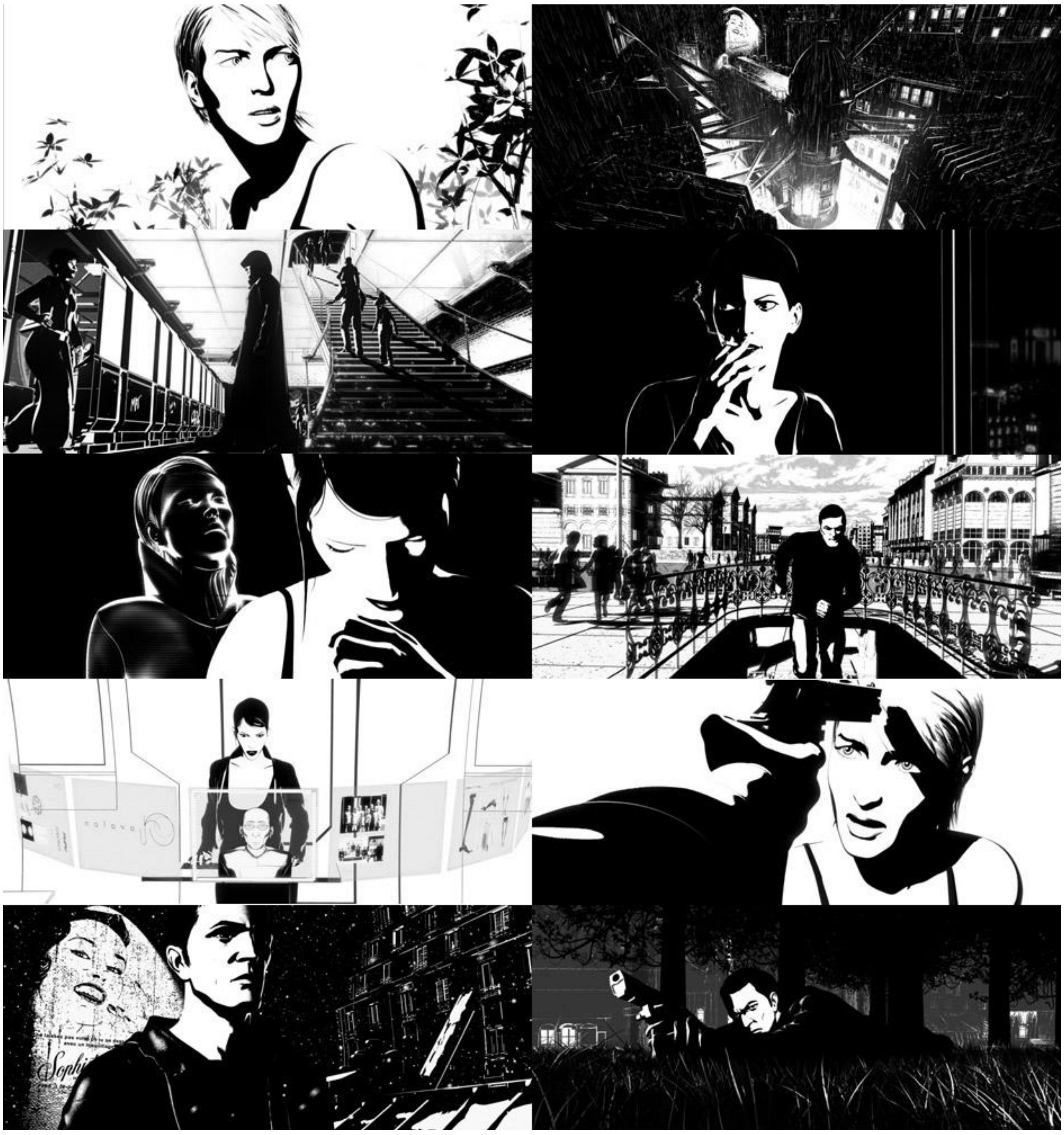
The full paper can be found here.

http://dspace.mit.edu/bitstream/handle/1721.1/61252/701866466.pdf (Miaw)

Motion Capture: Is it just a clever gimmick?

I have been fascinated by Motion Capture ever since I first heard of it but not everybody feels this way. My partner says if someone is going to act she would rather see their real face and not put the many technicians and crew involved in a film out of a job.

The counter argument to this is something like Renaissance, the 2004 French CGI film shot entirely in solid black and solid white (monochrome) to look like a comic book which achieved a stylised effect of a futuristic Paris which would not have been possible without the use of this technique.



The sheer "Mask"-like possibility of putting on any CGI character and acting through this is attractive to me and many films have used it to good effect.

The influence on the computer games industry has been profound. Since Space Invaders, Virtual Reality has come a really long way and games today are just starting to access the sort of photorealism that users deserve. It has been a huge investment by gamers from every slot machine and copy sold into the technology which will make the future of VR worlds a reality.

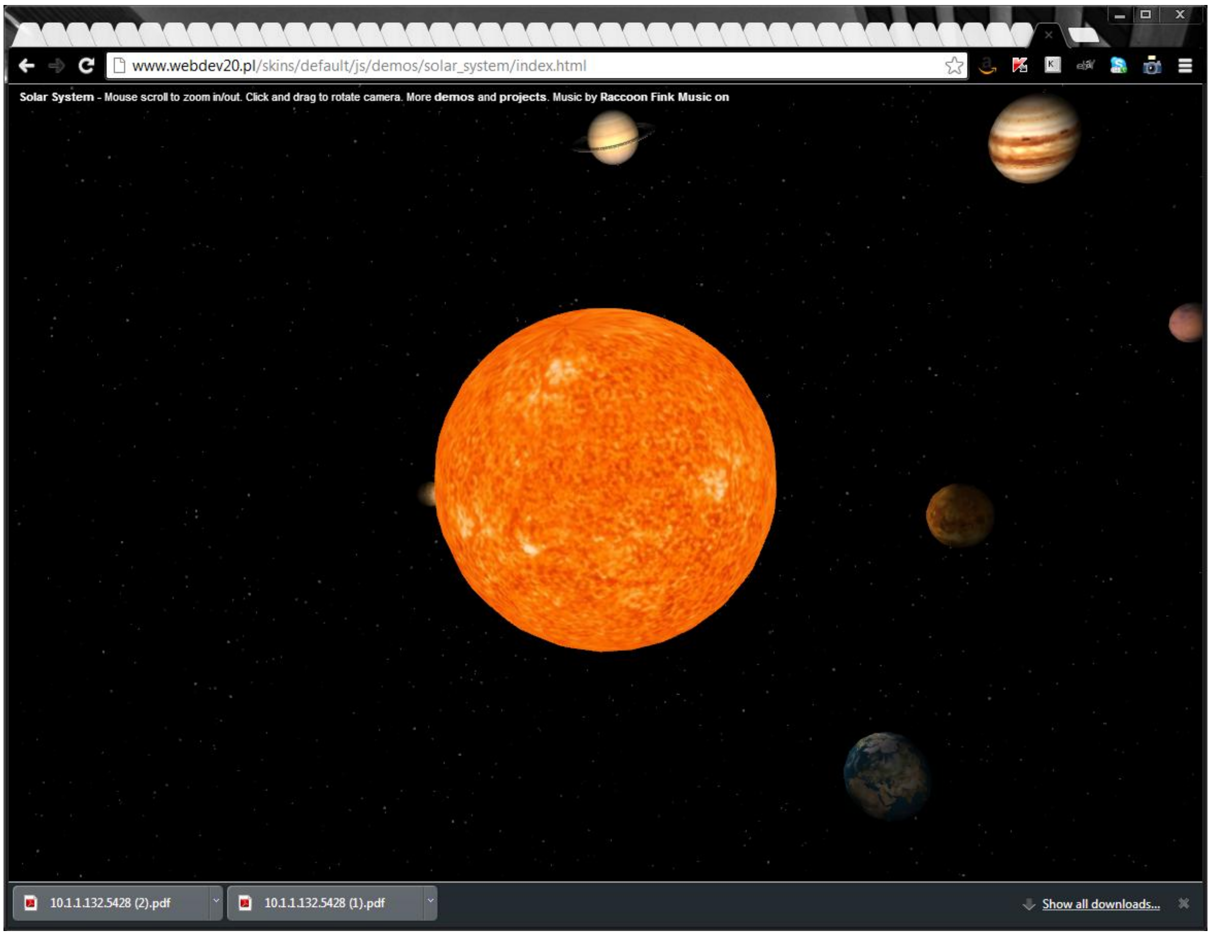
Gaming now represents a bigger market than film and music put together so could be viewed as one of the dominant art-forms of our culture.

The possibility of augmented reality and immersive worlds powered by mobile technology opens up new possibilities of interactive gaming and I wonder how long it will be before “holodecks” start appearing in amusement arcades.

Of course MoCap has technical uses in industry and science and we have seen earlier an application in special needs education.

Virtual Reality too is becoming more available with the introduction of WebGL (a pared down version of OpenGL) allowing 3d models to be displayed directly by web browsers.

Early Chrome demos can be seen at http://www.chromeexperiments.com/webgl/ (Chrome Experiements)


www.webdev20.pl/skins/default/js/demos/solar_system/index.html
Solar System - Mouse scroll to zoom in/out. Click and drag to rotate camera. More demos and projects. Music by Raccoon Fink Music on
10.1.1.132.5428 (2).pdf
10.1.1.132.5428 (1).pdf
Show all downloads...

http://www.webdev20.pl/skins/default/js/demos/solar_system/index.html (webdev20.pl)

All in all it seems like Virtual reality is a technology whose time has come with the standard gaming Video Card now powerful enough to do reasonable quality rendering.

With model making sites like http://www.turbosquid.com, http://daz3d.com selling completed models there is a lot of interest in the subject and it seems the only thing holding people back is the expense of the software – this is something that better publicity for Blender could change.

Bibliography

Brown University. (n.d.). *3D Photography and Geometry Processing.* Retrieved from http://mesh.brown.edu/3DPGP-2009/homework/hw2/hw2.html

Brown University. (n.d.). *3d with OpenGL.* Retrieved from http://www.cs.brown.edu/courses/cs123/lectures/03_GL_3D.pptx

Chrome Experiements. (n.d.). *WebGL experiements.* Retrieved from http://www.chromeexperiments.com/webgl/

David Lasercanner. (n.d.). Retrieved from http://www.david-laserscanner.com/

Lee, J. (n.d.). *Hybrid Infrared and Visible Light Projection.* Retrieved from http://johnnylee.net/projects/thesis/

Miaw, D. (n.d.). *Second Skin: Motion Capture with Actuated Feedback for Motor Learning.* Retrieved from http://dspace.mit.edu/bitstream/handle/1721.1/61252/701866466.pdf

Moderbacher, S. (n.d.). *Student Work.* Retrieved from http://students.autodesk.com/?nd=showcase_detail_page&gallery_id=18518&jid=191413

Page, R. L. (n.d.). *Breif History Of Flight Simulation*. Retrieved from http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.132.5428&rep=rep1&type=pdf

Polygon Puppet. (n.d.). *Female anatomy.* Retrieved from Turbosquid: http://www.turbosquid.com/3d-models/3d-female-circulatory-heart/694548

Reddy, M. (n.d.). *.obj specification.* Retrieved from http://www.martinreddy.net/gfx/3d/OBJ.spec

Sabry, K. (n.d.). Retrieved from http://students.autodesk.com/?nd=showcase_detail_page&gallery_id=18516&jid=191413

ScanLAB. (n.d.). *The Angel Building, AHMM Architects.* Retrieved from http://vimeo.com/30737412

Stanford University. (n.d.). *Markerless Motion Capture Project.* Retrieved from https://ccrma.stanford.edu/~stefanoc/Markerless/Markerless.html

Tim & Joe. (n.d.). *New Look - Nap On The Bow.* Retrieved from http://vimeo.com/30427082

UCSF. (n.d.). *AmberTools12 Reference Manual*. Retrieved from University of Callifornia at San Francsico Amber Molecular Dynamics Project: http://ambermd.org/doc12/AmberTools12.pdf

Watson, T. (n.d.). *Interactive Puppet Prototype with Xbox Kinect.* Retrieved from http://vimeo.com/16985224

webdev20.pl. (n.d.). *Solar System.* Retrieved from http://www.webdev20.pl/skins/default/js/demos/solar_system/index.html

wikipedia. (n.d.). Retrieved from http://en.wikipedia.org/wiki/Parallax

Wikipedia. (n.d.). *Distortion (optics).* Retrieved from http://en.wikipedia.org/wiki/Distortion_(optics)

Wikipedia. (n.d.). *Pulse Width Modulation.* Retrieved from wikipedia.org: http://en.wikipedia.org/wiki/Pulse-width_modulation

Appendix 1: Renderer in VB.net (Alex Hughes)

```
Imports System.IO

Public Class Form1

    Dim StreamToDisplay As StreamReader




    Private Sub Form1_load(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles MyBase.Load

    End Sub

    Private Sub btnOpen_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnOpen.Click

        OpenFileDialog1.Filter = "Object Files (*.obj)|*.obj"
        If OpenFileDialog1.ShowDialog() = DialogResult.OK Then

            StreamToDisplay =
My.Computer.FileSystem.OpenTextFileReader(OpenFileDialog1.FileName)

            ReadPreamble()
            ReadName()
            ReadVertices()
            ReadFaces()

            StreamToDisplay.Close()


        End If
    End Sub

    Private Sub ReadPreamble()
        Do
            LineOfText = StreamToDisplay.ReadLine()
            strAllText = strAllText & LineOfText & vbCrLf
        Loop Until LineOfText = ""

    End Sub
    Private Sub ReadName()
        Do
            LineOfText = StreamToDisplay.ReadLine()
            strAllText = strAllText & LineOfText & vbCrLf
        Loop Until LineOfText = ""

    End Sub

    Private Sub ReadVertices()
        intVertexNo = 0


        Do


            LineOfText = StreamToDisplay.ReadLine()
            strVert = Split(LineOfText, " ", 5)
            If strVert(0) = "v" Then
```

```
                intVertexNo = intVertexNo + 1
                ReDim Preserve decVertices(2, intVertexNo)
                For nl As Integer = 0 To 4
                    strAllText = strAllText & strVert(nl) & " "
                Next nl
                strAllText = strAllText & vbCrLf
                For nl As Integer = 0 To 2
                    decVertices(nl, intVertexNo) = CDec(strVert(nl + 2))
                Next nl


            ElseIf strVert(0) = "#" Then
                strAllText = strAllText & strVert(1) & vbCrLf
                strAllText = strAllText & intVertexNo & vbCrLf
            End If
        Loop Until LineOfText = ""

    End Sub

    Private Sub ReadFaces()
        intFaceNo = 0
        Do


            LineOfText = StreamToDisplay.ReadLine()
            strFace = Split(LineOfText, " ", 5)
            If strFace(0) = "f" Then
                intFaceNo = intFaceNo + 1
                ReDim Preserve intFaces(2, intFaceNo)
                For nl As Integer = 0 To 3
                    strAllText = strAllText & strFace(nl) & " "
                Next nl
                strAllText = strAllText & vbCrLf
                For nl As Integer = 0 To 2
                    intFaces(nl, intFaceNo) = CDec(strFace(nl + 1))
                    strAllText = strAllText & intFaces(nl, intFaceNo)
                Next nl


            ElseIf strFace(0) = "#" Then
                strAllText = strAllText & strFace(1) & vbCrLf
                strAllText = strAllText & intFaceNo & vbCrLf
            End If
        Loop Until LineOfText = ""

    End Sub

    Private Sub PrintArrays()
        strAllText = intVertexNo & " vertices" & vbCrLf
        For nc As Integer = 1 To intVertexNo
            For nl As Integer = 0 To 2
                strAllText = strAllText & decVertices(nl, nc) & " "
            Next nl
            strAllText = strAllText & vbCrLf
        Next nc
        strAllText = strAllText & vbCrLf & intFaceNo & " faces" & vbCrLf

        For nc As Integer = 1 To intFaceNo
            For nl As Integer = 0 To 2
                strAllText = strAllText & intFaces(nl, nc) & " "
            Next nl
            strAllText = strAllText & vbCrLf
        Next nc
        '  txtObject.Text = strAllText
    End Sub

    Private Sub RenderObjectWireframe()
        Dim zVertex(intFaceNo) As Decimal
        For nc As Integer = 1 To intFaceNo
            zVertex(nc) = 0

            For nl As Integer = 0 To 2
                zVertex(nc) = decVertices(2, intFaces(0, nc))
            Next nl
        Next nc
        ' Array.Sort(zVertex, intFaces)
        Dim FrontBackFace(intFaceNo) As Boolean
        Dim Normal(2, intFaceNo)
        For nc As Integer = 1 To intFaceNo

        Next

        Dim scale As Decimal = 5
        Dim lens As Decimal = 100
        Dim Zoff As Decimal = 10000
        Dim Xoff As Integer = -1500
        Dim Yoff As Integer = 500
        decVertices(0, 0) = 0
        decVertices(1, 0) = 0
        decVertices(2, 0) = -Zoff / scale

        Dim PenColor As New Pen(Color.Green)
        Dim x(2) As Decimal
        Dim y(2) As Decimal
        Dim z(2) As Decimal
        GraphicsFun = Me.CreateGraphics


        For nc As Integer = 1 To intFaceNo
            lblMessage.Text = nc



            For nl As Integer = 0 To 2
                x(nl) = decVertices(0, intFaces(nl, nc))
                y(nl) = decVertices(1, intFaces(nl, nc))
                z(nl) = decVertices(2, intFaces(nl, nc))
            Next nl

            Dim sX(2) As Integer
            Dim sY(2) As Integer

            For nl As Integer = 0 To 2
                sX(nl) = CInt(Xoff + (x(nl) * scale * lens / (z(nl) * scale
+ Zoff)))
                sY(nl) = CInt(Yoff - (y(nl) * scale * lens / (z(nl) * scale
+ Zoff)))
            Next nl
            FrontBackFace(nc) = sX(0) * sY(1) > sY(0) * sX(1)
            Dim Points() As Point = {New Point(sX(0), sY(0)), New
Point(sX(1), sY(1)), New Point(sX(2), sY(2))}
            GraphicsFun.DrawPolygon(PenColor, Points)
        Next nc



    End Sub
    Private Sub RenderObjectFlat()
        Dim scale As Decimal = 25
        Dim lens As Decimal = 3000
        Dim Zoff As Decimal = 5000
        Dim Xoff As Integer = 500
        Dim Yoff As Integer = 500
        decVertices(0, 0) = 0
        decVertices(1, 0) = 0
        decVertices(2, 0) = -Zoff / scale
        intFaces(0, 0) = 0
        intFaces(1, 0) = 0
        intFaces(2, 0) = 0
        Dim intShadeValue(intFaceNo) As Integer
        Dim colShade(intFaceNo) As Color

        Dim decLight() As Decimal = {0.5, 0.5, -0.5}
        Dim decIntensity As Decimal = 130
        Dim decAmbient As Decimal = 40
        ReDim decVerticesScreen(1, intVertexNo)
        Dim Points(intVertexNo) As Point

        For nl As Integer = 1 To intVertexNo
            decVerticesScreen(0, nl) = Xoff + (decVertices(0, nl) * scale *
lens / (decVertices(2, (nl)) * scale + Zoff))
            decVerticesScreen(1, nl) = Yoff + (decVertices(1, nl) * scale *
lens / (decVertices(2, (nl)) * scale + Zoff))
            Points(nl) = New Point(CInt(decVerticesScreen(0, nl)),
CInt(decVerticesScreen(1, nl)))
        Next nl

        Dim zVertex(intFaceNo) As Decimal
        Dim intFaceOrder(intFaceNo) As Integer

        For nc As Integer = 0 To intFaceNo

            zVertex(nc) = 0
            For nl As Integer = 0 To 2
                zVertex(nc) = zVertex(nc) + decVertices(2, intFaces(nl,
nc)) / 3
            Next nl

            intFaceOrder(nc) = nc
        Next nc
        Array.Sort(zVertex, intFaceOrder)




        Dim ClockAnticlock(intFaceNo) As Integer
        Dim Normal(2, intFaceNo) As Decimal
        Dim decEdge0(2, intFaceNo) As Decimal
        Dim decEdge1(2, intFaceNo) As Decimal
        Dim decEdge0Screen(1, intFaceNo) As Decimal
        Dim decEdge1Screen(1, intFaceNo) As Decimal


        Dim red As Decimal = Rnd()
        Dim green As Decimal = Rnd()
        Dim blue As Decimal = Rnd()
        Dim tot As Decimal = red + green + blue
        red = red / tot
        green = green / tot
        blue = blue / tot
        For nc As Integer = 1 To intFaceNo
            For ni As Integer = 0 To 2
                decEdge0(ni, nc) = decVertices(ni, intFaces(1, nc)) -
decVertices(ni, intFaces(0, nc))
                decEdge1(ni, nc) = decVertices(ni, intFaces(2, nc)) -
decVertices(ni, intFaces(1, nc))
            Next ni

            For ni As Integer = 0 To 1
                decEdge0Screen(ni, nc) = decVerticesScreen(ni, intFaces(1,
nc)) - decVerticesScreen(ni, intFaces(0, nc))
                decEdge1Screen(ni, nc) = decVerticesScreen(ni, intFaces(2,
nc)) - decVerticesScreen(ni, intFaces(1, nc))
            Next ni

            If decEdge0Screen(0, nc) * decEdge1Screen(1, nc) <
decEdge0Screen(1, nc) * decEdge1Screen(0, nc) Then
                ClockAnticlock(nc) = 1
            Else
                ClockAnticlock(nc) = +1
            End If
            Normal(0, nc) = decEdge0(1, nc) * decEdge1(2, nc) - decEdge0(2,
nc) * decEdge1(1, nc)
            Normal(1, nc) = decEdge0(2, nc) * decEdge1(0, nc) - decEdge0(0,
nc) * decEdge1(2, nc)
            Normal(2, nc) = decEdge0(0, nc) * decEdge1(1, nc) - decEdge0(1,
nc) * decEdge1(0, nc)


            Dim lenNormal As Decimal = Math.Sqrt(Normal(0, nc) * Normal(0,
nc) + Normal(1, nc) * Normal(1, nc) + Normal(2, nc) * Normal(2, nc))
            If lenNormal = 0 Then lenNormal = 1

            For ni As Integer = 0 To 2
                Normal(ni, nc) = Normal(ni, nc) * ClockAnticlock(nc) /
lenNormal
            Next ni


            Dim NormalLightCos As Decimal = ((Normal(0, nc) * decLight(0))
+ (Normal(1, nc) * decLight(1)) + (Normal(2, nc) * decLight(2)))
            If NormalLightCos < 0 Then
                NormalLightCos = 0
            End If
            intShadeValue(nc) = CInt((NormalLightCos * decIntensity) +
decAmbient)




            'colShade(nc) = Color.FromArgb(255, CInt(intShadeValue(nc) *
red), CInt(intShadeValue(nc) * green), CInt(intShadeValue(nc) * blue))
            colShade(nc) = Color.FromArgb(255, intShadeValue(nc),
intShadeValue(nc), intShadeValue(nc))
        Next nc


            For nc As Integer = intFaceNo To 0 Step -1
                If intFaceOrder(nc) = 0 Then Exit For
                Dim PenColor As New Pen(Color.Green)
                Dim PointsToDraw() As Point = {Points(intFaces(0,
intFaceOrder(nc))), Points(intFaces(1, intFaceOrder(nc))),
Points(intFaces(2, intFaceOrder(nc)))}
                GraphicsFun = Me.CreateGraphics

            Dim brushcolor As New SolidBrush(colShade(intFaceOrder(nc)))
            GraphicsFun.FillPolygon(brushcolor, PointsToDraw)
            'GraphicsFun.DrawPolygon(PenColor, PointsToDraw)


            Next nc


    End Sub

    Private Sub Form1_Paint(ByVal sender As Object, ByVal e As
System.Windows.Forms.PaintEventArgs) Handles Me.Paint
        RenderObjectFlat()


    End Sub

    Private Sub btnRLeft_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRLeft.Click
        Dim Cosangle = Math.Cos(angle)
        Dim Sinangle = Math.Sin(angle)
        For nc As Integer = 1 To intVertexNo
            oX = decVertices(0, nc)
            oY = decVertices(1, nc)
            oZ = decVertices(2, nc)

            oXnew = oX * Cosangle - oZ * Sinangle
            oZnew = oZ * Cosangle + oX * Sinangle


            decVertices(0, nc) = oXnew
            decVertices(2, nc) = oZnew
        Next nc
        Dim brushcolor As New SolidBrush(Color.White)
        GraphicsFun.FillRectangle(brushcolor, 0, 0, 1000, 1000)
        RenderObjectFlat()


    End Sub
    Private Sub btnRUp_Click(ByVal sender As System.Object, ByVal e As
System.EventArgs) Handles btnRUp.Click
        Dim Cosangle = Math.Cos(angle)
        Dim Sinangle = Math.Sin(angle)
        For nc As Integer = 1 To intVertexNo
            oX = decVertices(0, nc)
            oY = decVertices(1, nc)
            oZ = decVertices(2, nc)

            oYnew = oY * Cosangle - oZ * Sinangle
            oZnew = oZ * Cosangle + oY * Sinangle


            decVertices(1, nc) = oYnew
            decVertices(2, nc) = oZnew
        Next nc
        Dim brushcolor As New SolidBrush(Color.White)
        GraphicsFun.FillRectangle(brushcolor, 0, 0, 1000, 1000)
        RenderObjectFlat()
    End Sub
End Class
```