# *Interactive Computer Graphics*

Lecture 17

Kinematics and Animation

## *Animation of 3D models*

In the early days physical models were altered frame by frame to create animation - eg King Kong 1933.

Computer support systems for animation began to appear in the late 1970, and the first computer generated 3D animated full length film was Toy Story (1995).

## *Computer Aids to Animation*

Fully automatic animation has not proved successful. However computer tools to support animation have developed rapidly.

These remove much of the tedious work of the animator, and allow the creation of spectacular special effects.
Basic Approaches are:

- Physical Models
- Procedural Methods
- Keyframing

## *Physical Modelling*

This approach is suitable for inanimate objects or effects where there is a simple physical model:

- Bouncing balls
- Cars on rough roads

Effects can be created by Newtonian mechanics

In some cases composite models can be applied

- Spring mass damper arrays for fluttering flags
- Particle systems for fire smoke etc.

## *Procedural Approach*

The behaviour of an object is described by a procedure, sometimes specified by a script.

This kind of approach is appropriate for well known behaviours in inanimate objects:

eg crack propagation in glass or concrete which is difficult to model physically but easy to describe procedurally.

## *Keyframe Approach*

Animators specify two positions of the skeleton (keyframes) a short time apart. The computer then calculates a number of “in between” positions to effect a smooth movement between the two keyframes.

This was a traditional method in hand drawn animation where the senior animator drew the key frames and some stooge would then laboriously draw all the in betweens.

## Creating In-betweens

The in-betweens are created by interpolation.

For certain objects they can be created by using paths of motion defined by, for example, spline curves.

However there are difficulties:

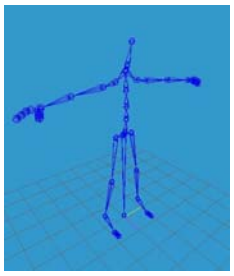
Observing the laws of physics
Making plausible movements

## Movement Control

When animating vital characters any model animation must be kept plausible. Hence computer systems are often based on jointed skeletons.

Each link in the articulated chain is rigid. The movement is constrained by the degree of freedom at each joint.

The skeleton can be fleshed out in any way.



## Luxo Jr. (1987)

This award winning short film was the hailed as the first example of computer generated 3D animation that was as natural as hand crafted animation.

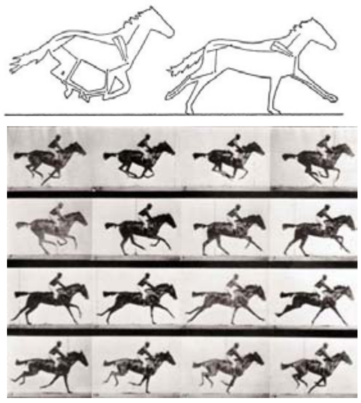
It was based on the use of an articulated skeleton.

http://www.pixar.com/shorts/ljr/theater/short_320.html

## Complexities of Natural Motion

Natural motions are complex, as shown by Eadweard Muybridge's horse photographs (1878).

Some systematic approach is needed for "scripting" the motion of the skeleton.



## Motion Capture

One approach to defining the movement is to use motion capture - for example Gollum in the Lord of the Rings films.

These methods require laborious data capture and are surprisingly hard to automate.

## Motion Capture

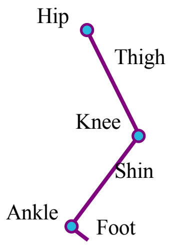
King Kong (2005) also used motion capture.

The actor wore a blue suit and was marked with around 80 identifiable spots that could be tracked by cameras.

The spots were mapped (not linearly) to corresponding points on the computer Kong model.

The actor had to study gorillas and then imitate their movements.

## Kinematics



Although natural skeletons are not strictly rigid, it is a good approximation to treat them as such and thus constrain the movement.

The study of the movement of articulated chains began in mechanical engineering of robots and is called kinematics.

## Degrees of freedom

Hinge Joint - One degree of freedom
Knee or elbow joint

Saddle - Two degrees of freedom
Wrist/Hand joints

Socket Joint - Three degrees of freedom
Hip, shoulder neck

Images from www.shockfamily.net

## Euler Angles

At any joint we can specify the orientation of one link relative to the other by the Euler angles. These encode pitch, roll and yaw.

The links are fixed in length (rigid), so the position of end of the chain is completely defined by a set of Euler angles.

For the leg we have 3(hip) + 1(knee) + 1 (ankle) = 5 variables (Euler Angles) to determine the end point.

## Euler Angles

Euler was the first mathematician to prove that any rotation of 3D space could be achieved by three independent rotations.

There is no standard way of achieving this, but the usual convention is to:

1. Rotate about Z
2. Rotate about X
3. Rotate about Z

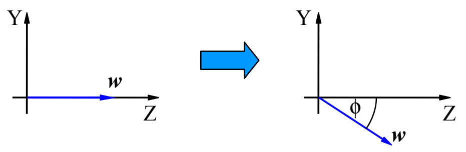
The complete Euler rotation matrix $R_E$ is therefore:

$$\begin{bmatrix} Cos(\theta) & Sin(\theta) & 0 & 0 \\ -Sin(\theta) & Cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & Cos(\phi) & Sin(\phi) & 0 \\ 0 & -Sin(\phi) & Cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} Cos(\psi) & Sin(\psi) & 0 & 0 \\ -Sin(\psi) & Cos(\psi) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Euler Angles

This formulation is not very intuitive, so to see what is happening consider transforming a vector ***w*** lying along the z axis.

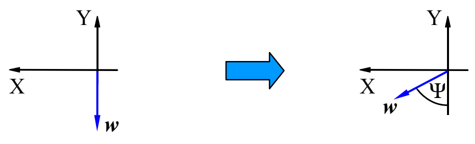
The first rotation will not change it at all, but the second can rotate it to any position in the *y-z* plane.



## Euler Angles

The last rotation about the *z* axis can make the projection of ***w*** point in any direction in the *x-y* plane.

Looking from the positive z axis we have:



So the last two rotations can orient ***w*** in any direction in the 3D space.

## *Euler Angles*

The first rotation turns the ***u*** and ***v*** directions about the ***w***. This would be equivalent to a roll, or the rotation of an image about its centre if ***w*** were the viewing direction.

Thus the specifying the three Euler angles allows us to rotate a (***u***,***v***,***w***) axis system to any orientation we require, while preserving its orthogonality.

## *Forward Kinematics*

Forward kinematics is used in robot control. Given a specification of the Euler angles of each joint or an articulated robot arm we calculate the position and orientation of its end point.

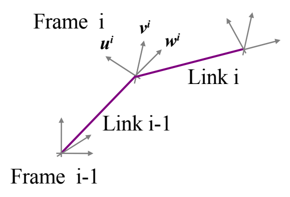
This is a well posed problem and can be solved by matrix transformations of space similar to the ones that we have already seen in use - for example in first person shoot 'em ups.

## *Forward Kinematics Computation*

We define a coordinate system at the start of a chain and at each joint.

Each new coordinate system is defined in the co-ordinate system of the previous joint.

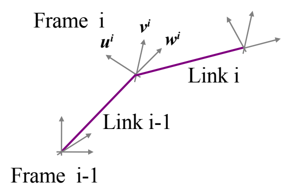
Thus the position $\mathbf{C}^i$ and the direction vectors $\{\boldsymbol{u}^i, \boldsymbol{v}^i, \boldsymbol{w}^i\}$ of frame i are defined using the frame i-1 coordinate system.



## *Forward Kinematics Computation*

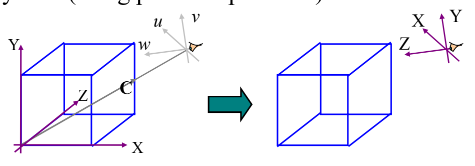
The position $\mathbf{C}^i$ and the direction vectors $\{\boldsymbol{u}^i, \boldsymbol{v}^i, \boldsymbol{w}^i\}$ of each frame can be calculated simply using the Euler rotation matrix $R_E$: $\boldsymbol{u}^i = [1,0,0]R_E$, $\boldsymbol{v}^i = [0,1,0]R_E$, $\boldsymbol{w}^i = [0,0,1]R_E$ and $u^i = [0,0,L_{i-1}]R_E$ where $L_{i-1}$ is the length of link (i-1).

We can express positions and directions in frame i in the coordinate system of frame i-1 by a matrix transformation, which is the inverse of the viewing transformation discussed earlier in the course.



## *Revision - the viewing transformation*

To transform the scene coordinates to the axis system $\{\boldsymbol{u},\boldsymbol{v},\boldsymbol{w}\}$ system (using pre-multiplication) we used:



$$[P^t_x, P^t_y, P^t_z, 1] = [P_x, P_y, P_z, 1]\begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ -C\cdot u & -C\cdot v & -C\cdot w & 1 \end{bmatrix}$$

## *Revision - Viewing transformation*

The inverse of the viewing transformation (which we need for forward kinematics) can be written as:

$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ C_x & C_y & C_z & 1 \end{bmatrix}$$

which can be verified by multiplying out:

$$\begin{bmatrix} u_x & u_y & u_z & 0 \\ v_x & v_y & v_z & 0 \\ w_x & w_y & w_z & 0 \\ C_x & C_y & C_z & 1 \end{bmatrix}\begin{bmatrix} u_x & v_x & w_x & 0 \\ u_y & v_y & w_y & 0 \\ u_z & v_z & w_z & 0 \\ -C\cdot u & -C\cdot v & -C\cdot w & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

## Forward Kinematics Computation

If we denote the transformation from the ith to the i-1th frame as:

$$T_i^{i-1} = \begin{bmatrix} u_x^i & u_y^i & u_z^i & 0 \\ v_x^i & v_y^i & v_z^i & 0 \\ w_x^i & w_y^i & w_z^i & 0 \\ C_x^i & C_y^i & C_z^i & 1 \end{bmatrix}$$

Then the complete forward transformation (using pre-multiplication) can be seen to be:

$$T_n^0 = \prod_{i=n}^{0} T_i^{i-1}$$

## Inverse Kinematics

For graphics applications we want the opposite process. We wish to specify some path for the end point of the chain, and calculate the relative positions of all the other links.

This is an ill posed problem (there are infinitely many solutions for some chains).

Hence we need to find a constrained solution minimising for example, the joint movements.

## Inverse Kinematics

In practice, inverse kinematics can be used to calculate in-between frames.

For example, to generate ten frames of a leg movement, we define the ten steps that make up the foot movement, and estimate the changes in the Euler angles of the rest of chain that implement those changes.

In the simple articulated leg chain there are five Euler angles. The constrained angles are set to zero.

## Inverse Kinematics by Gradient Descent

One way to solve the problem is to use gradient descent.

Let E be the distance between the end point and its target.

For each Euler angle $\theta$ we find $dE/d\theta$ using forward kinematics, replacing $\theta$ with $\theta+\Delta\theta$ and calculating $\Delta E$.

We then update the angles using:

$$\theta^t = \theta^{t-1} - \mu\, dE/d\theta$$

## Further Constraints

Applying gradient descent with small steps should find a solution that involves only small joint movements.

However, over time it is possible that a skeleton will reach an implausible pose.

One solution is to constrain the optimisation to avoid poor poses. Poor pose is difficult to define, and needs analysis of videos of actual motion.

## In Summary

Currently human intervention is a necessary part of creating realistic living model animation. Either motion capture or expert animator input.

Physical modelling and procedural methods are successful in creating movements in inanimate objects.

Inverse kinematics can take some of the burden out of living model animation, but is limited and is an interesting current research area.

## *Interactive Graphics Lecture 18*

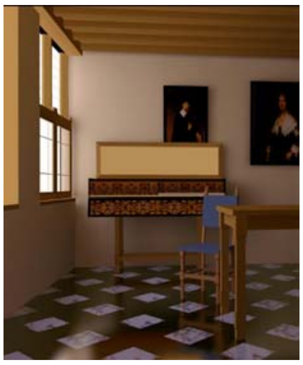
Non-Photorealistic Rendering

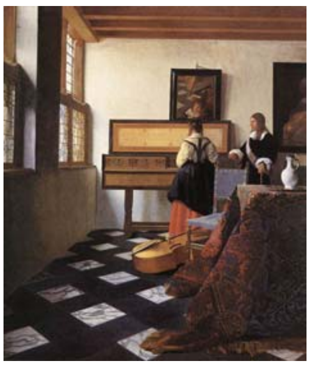
## *Why Photorealistic?*

Much graphics research is aimed at producing photo-realism. Techniques that we have discussed include:

- textures
- bump mapping
- environment mapping
- ray tracing
- radiosity

Modern research continues this quest.


Photorealistic Rendering
Cornell University
circa 1990

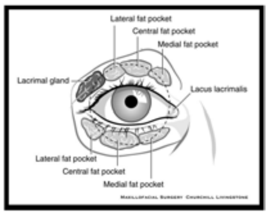

Non-photorealistic Rendering
Johannes Vermeer
circa 1660

## *Why Non-Photorealistic?*

Schematic diagrams can show things that photos can't

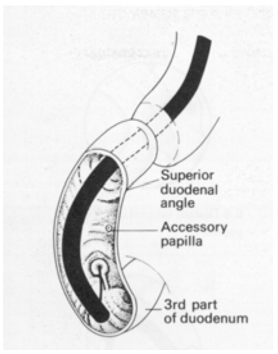
An artist is often able to (expected to) convey "expressiveness".

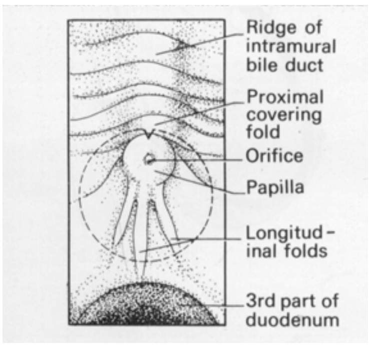
This has given rise to the field of non-photorealistic rendering



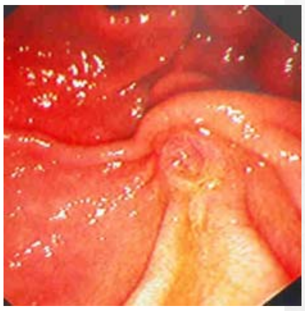
Medical illustration
From IBLS at University of Glasgow

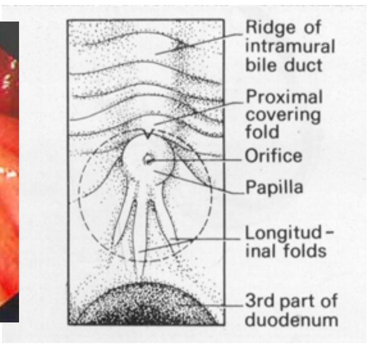
## *Non-Photorealistic medical illustrations*





## *Non-Photorealistic medical illustrations*





## *Styles and Effects for media and games*

An important commercial application for NPR is creating special “looks” for games or cartoon films. This can be done by:

Special rendering effects - either based on polygon rendering or on ray tracing

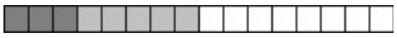
Post processing films or cartoons. This has proved hard to achieve as NPR often doesn't work well on a range of images. Hand processing of single frames is too time consuming.

**Cell-Shading** – Rendering a 3D polygon scene as a cartoon. Used in video games (dreamcast)

SegaWeb.Com

## *Cell Shading - uses polygon rendering*

Create a Light map: this is a 1-Dimensional texture map that indicates the shade of an object. It is set up using a few discrete regions.



Find a reflectance with Lambert's cosine law, use its value to select from the light map

Add black lines at the visible polygon boundaries (or some of them, eg occluding boundaries)

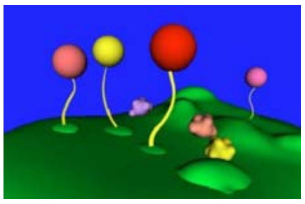
## *Creating other "animator" effects through rendering*

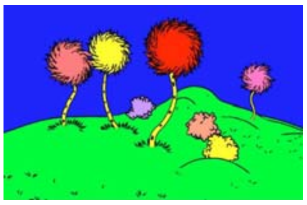
Remove shading

Outline strong edges

Add Embellishments

Work by Kowalski et al





## *Many NPR Systems Use Image filters*

Start with a photograph

**Blur - using the same techniques discussed for anti-aliasing**

**Quantise - Change colour or spatial resolution**

**Texture - combine texture and image by blending**

**Composite Filtering in different resolutions (more interesting)**

## *Blurring is done using the anti-aliasing filter*

Replace each pixel by a weighted average of its neighbourhood:

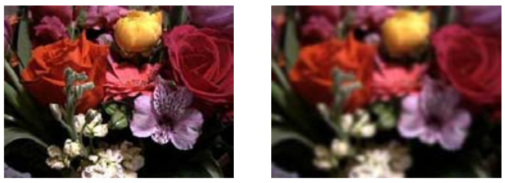
| | | |
|---|---|---|
| **1/36** | **1/9** | **1/36** |
| **1/9** | **4/9** | **1/9** |
| **1/36** | **1/9** | **1/36** |

Use several applications for more blurring, or use a larger filter kernel

## Blurring

Blurring images is fast and simple, but it doesn't really produce very interesting results.

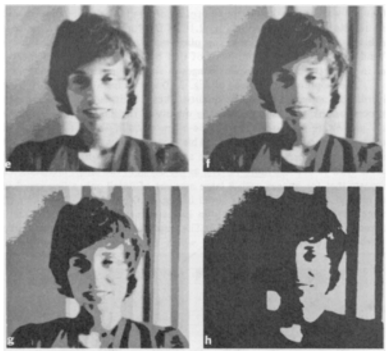
It is important in combination with other filters



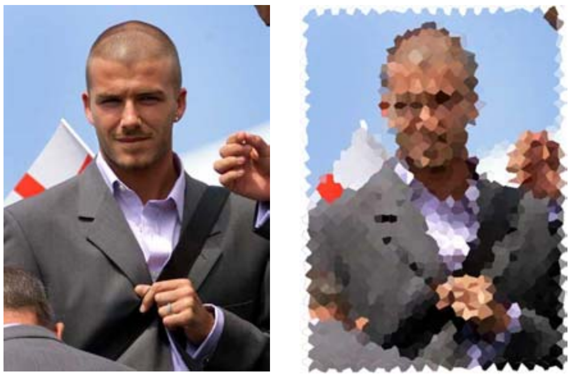
## Quantising Colour Depth

Change the number of colours or grey levels used to represent a picture.

Can produce interesting effects



## Quantising resolution and colour depth



## Edge Enhancement

Like blurring but with a different filter

Vertical Edge

| | | |
|---|---|---|
| **-1** | **0** | **1** |
| **-2** | **0** | **2** |
| **-1** | **0** | **1** |

Horizontal Edge

| | | |
|---|---|---|
| **1** | **2** | **1** |
| **0** | **0** | **0** |
| **-1** | **-2** | **-1** |

Find the magnitude of the two components

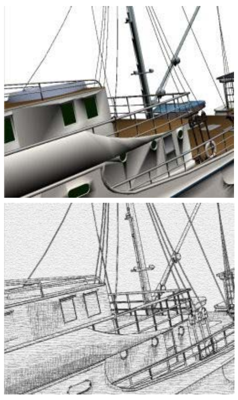
Change image depending on edge strength

## Creating pen images

Edges are found and reinforced

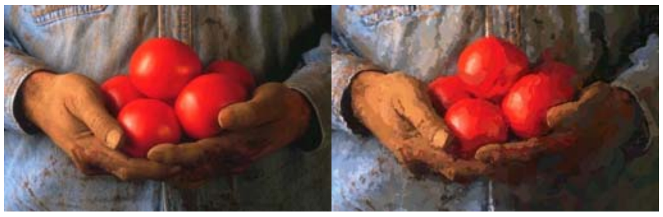
Shading is replaced with textures

Pictures from Intel 3D Software Technologies pages



## Composite filtering

Filtering in more complex manners can produce oil paint effects (Hertzmann and Perlin see http://www.mrl.nyu.edu/projects/npr/painterly/)



## Outline Algorithm

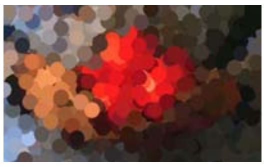
Initialise the output image to blank

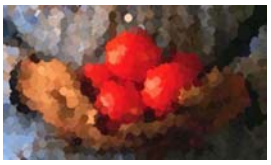
For a given brush size (eg 32,16, 8 or 4 pixels)

- Blur the source image using a filter size comparable to the brush size.
- Find a difference image between the blurred source image and the current output image.
- Threshold the difference image, so only large changes are retained.
- Find local maxima on a coarse grid
- Place a brush stroke on the output image at the local maximum using the corresponding image colour

## First stages of the technique using circular brush strokes

Source Image

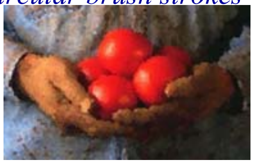

1. Circular brush strokes, radius 16

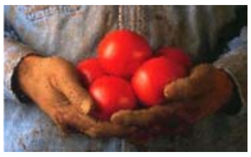

2. Circular brush strokes, radius 8

Images Angela Phuong IC 2006

## Refinement using small circular brush strokes

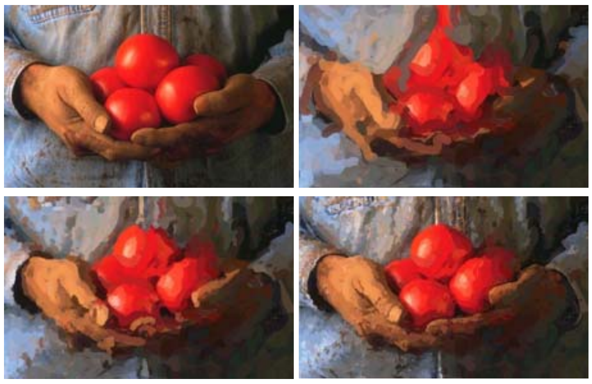
Source Image


3. Circular brush strokes, radius 4

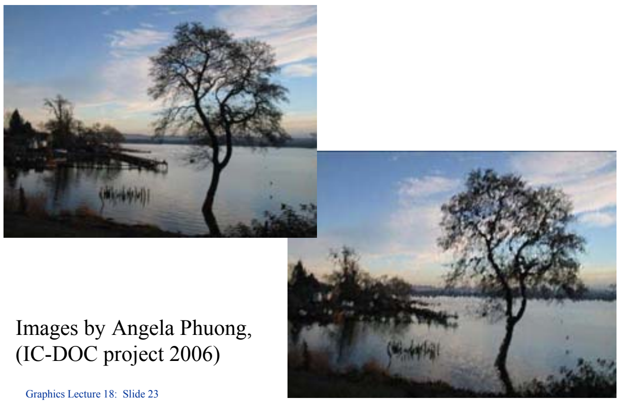

4. Circular brush strokes, radius 2

Images Angela Phuong IC 2006

## Hertzmann's original using elongated strokes



## A lakeside scene using circular brush strokes



Images by Angela Phuong, (IC-DOC project 2006)

## Creating effects by analogy

Work done by Hertzman Jacobs Oliver Curless and Salesin, University of Torronto (SIGGRAPH 2001)

The idea is to use effects created for one image on another image. Done using multi-resolution representation with local searches to find the best match.

The following examples are from:
http://www.mrl.nyu.edu/publications/image-analogies/

## Creating an analogy image

Given:

A: The original source image.

A': The original filtered image.

B: A new image to be filtered.

For each pixel of the target image:

Search for the best matching corresponding pair in A, A'

A naive search does not work well. We need to have matching metrics that take into account the coherence of the evolving image.

## Creating Image Analogies

The algorithm uses a multiscale approach, working from coarse to fine representations.

At each stage the search for the best match takes into account the local areas and the pixels in the previous level of the pyramid.

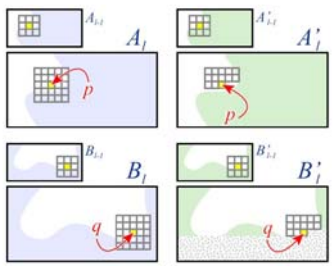


Image from Hertzmann et al.

Original Image

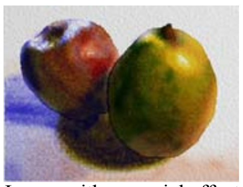
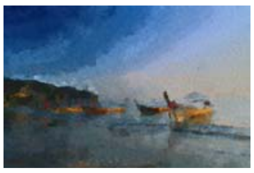
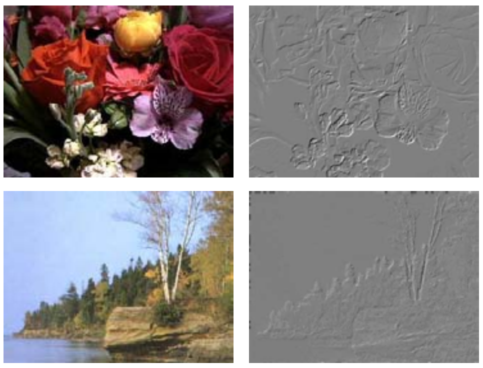


Image with a special effect

Take another image

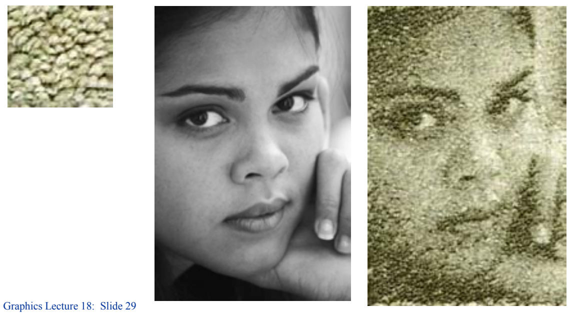

Create an analogous effect

## Analogies of basic filters -eg embossing



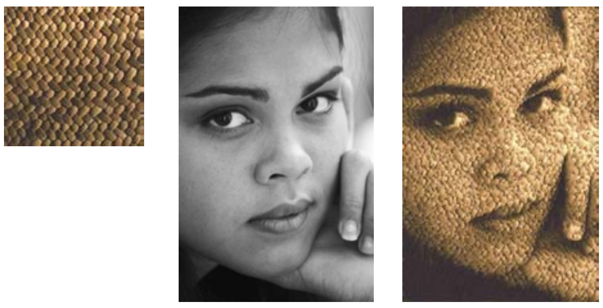
## Textures - mapped by analogy

Blending textures into images can create interesting results - The texture is used for A and A'



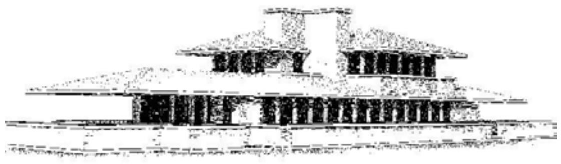
## Textures

Different textures create different effects



## *Rendering Methods*

Non photorealistic rendering can be applied directly to polygon maps



## *Creating pencil drawing effects*

Pencil drawings are made up from “strokes” of the pencil which have:

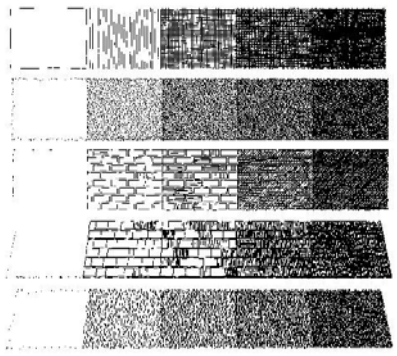
variation in width from pressure and direction

Tones (gradations of shade) and textures are created by

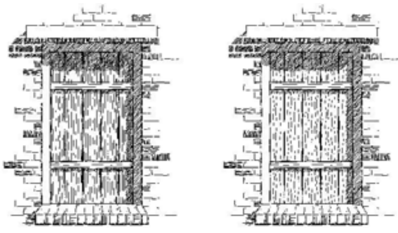
Combining strokes in different patterns

## *Different textures and tones*

Textures and tones can be pre-computed then used for rendering.



## *Effects of different strokes, tones and textures*



Polygon rendering with stroke textures

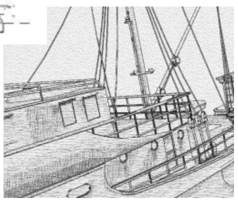


Image filtering with stroke textures

## *Stroke Textures*

Collection of strokes to give texture and tone

Prioritised so that different tones can be achieved

- first only highest priority drawn
- to increase tone, lower priorities drawn

For example:

- highest priority to outline
- next could be horizontal lines
- then vertical, and so on

## *Indication*

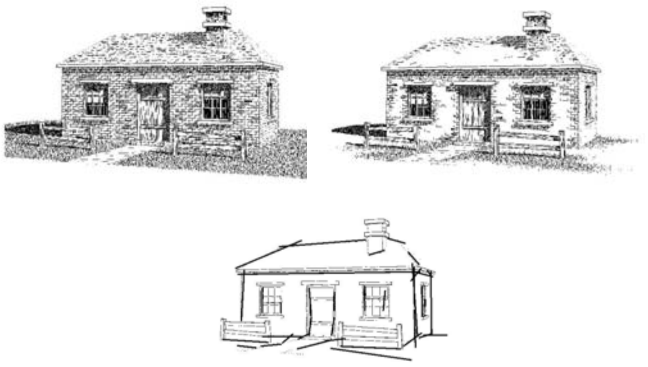
Texturing uniform areas uniformly does not produce good results.

Indication is the process of adding guidelines for texturing

The distance of a pixel from the guide line indicates the amount of texture used

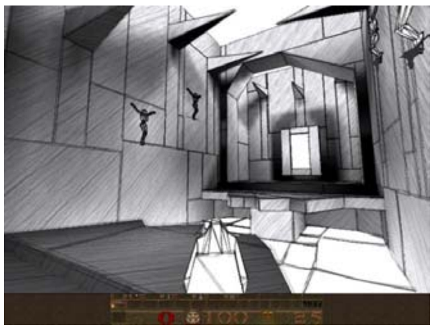
(Could be defined in the graphics scene)

## *Using Indication*



## *Pencil Sketch - Quake*

Main polygons shown - simple line shading - polygon edges rendered with stroke textures



# Tutorial 9:

# Animations, Transformations, Projections and Normalisation

This tutorial is a little out of place, but it will serve for revision.

1. *Animating Objects*

A cube in a graphics scene is defined by six square polygons. It is placed in the scene with its centre at (5, 5, 10). In an animation sequence it shrinks by 1/100th of its size in each successive frame until it is too small to be seen. Determine the transformation matrix, which when applied to the co-ordinates of the vertices of the cube will achieve the shrinkage required in each frame.

2. *Viewing transformations*

In a viewer centred animation (*Translator's note*: this means first person shoot 'em up), the viewer is at the point (10, 10, 10). The direction of view is **w** = $(0.6, -0.2, 0.77)^T$. The horizontal direction to the right is **u** = $(0.79, 0, -0.61)^T$.

Find the third unit vector **v** making up the axis system. Your result for **v** should point mainly upwards so if the *y* component you calculate is negative you have got the cross product the wrong way round.

Hence write down the viewing transformation matrix. If you've forgotten how to do this look in lecture notes on Scene Transformation and Animation, near the end.

3. *Projection*

The scene from the previous question is to be drawn in perspective projection on the plane $z = 2$. Find the required perspective projection matrix, and combined transformation and projection matrix.

A vertex of the scene has coordinate (10, 10, 20, 1). Where does it project to?

4. *Normalisation*

The world coordinate window in the plane $z = 2$ is between $(x, y) = (-5,-5)$ and $(x, y) = (5,5)$.

The window on the computer screen is of resolution 100 by 100 pixels, and the origin is in the top right hand side (bitmap organisation).

What is the pixel address of the vertex that was projected in the previous question?