An Introduction to 3D Computer Graphics, Stereoscopic Image, and Animation in OpenGL and C/C++

Fore June

Appendix B Video Compression using Graphics Models

B.1 Introduction

In the past two decades, a new development in video compression is to utilize synthesized images to represent some natural scenes that do not change much for a substantial period of time. For example, in a TV news announcement like the one shown in Figure B-1, the major scene change is the lip movement of the announcer. The background of the scene is fixed and in many cases may not be very important. This is also true for the situation of a video conference.

Figure B-1 News Announcer

To compress this kind of frames, one can use a graphics model to synthesize the announcer; the encoder just has to send the parameters of the lip movements of the announcer. Upon receiving the parameters, the decoder reconstructs the scene using the parameters and the graphics model. Of course, the parameters can be compressed before sending. One can easily see that this can give a very high compression ratio of the video as the encoder does not need to send the details of the images. It only has to send some parameters that will be used by the graphics model of the decoder to reconstruct the scene. The advance in computer graphics software and hardware technologies will make the use of graphics techniques play a more important role in video compression. The recent success of real 3D movies such as “Avatar” and “Alice and Wonderland” will further accelerate this trend.

In the 1990s, MPEG began to integrate synthetic audio and video data into audio and visual samples acquired from the natural world. The MPEG Synthetic-Natural Hybrid Coding was based on technologies developed by the Virtual Reality Modeling Language (VRML) Consortium (now Web3D). In later versions of MPEG-4 International Standard, efficient coding of shape and animation of human faces and bodies is specified. The specifications include standardizing Facial Animation (FA) and Face and Body Animation (FBA). MPEG-4 Visual specifies support for animated face and body models within the Simple Face Animation and simple FBA. A face model described by Facial Definition Parameters (FDPs) and animated using Facial Animation Parameters (FAPs) is specified. The body is described by body animation parameters (BAP). The basic idea of these technologies is to use graphics models to create synthesized human bodies or faces which are modified or deformed by parameters extracted from the real scene. Figure B-2 shows an example of such a model. (The image is taken from *http://coven.lancs.ac.uk/mpeg4/* .).

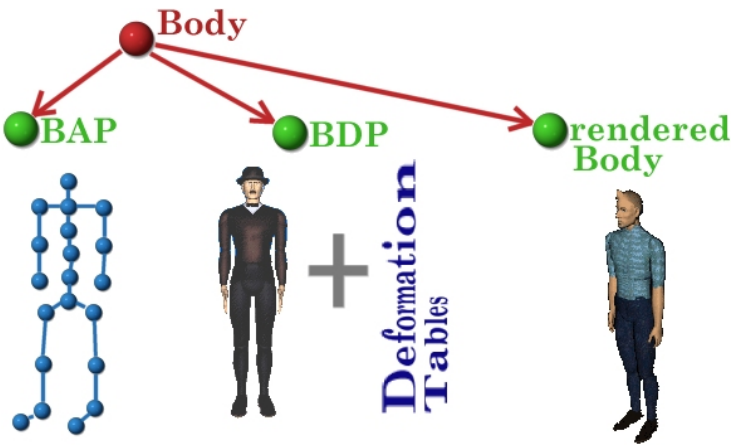


Figure B-2 MPEG-4 Body Animation

The next question is: *how do we generate synthesized human bodies or in general graphics objects?* It turns out that most 3D objects can be described using polygon meshes. Polygon meshes (or simply meshes) are collections of polygons that fit together to form the skin of the object. They have become a typical way of representing a broad class of solid graphical shapes. The simplest polygon is the triangle, which is one of the most useful geometric figures. The triangle is commonly used in the construction of buildings and bridges as it is the strongest geometric figure. Moreover, the vertices of a triangle always lie on the same plane. Therefore, it is the most popular polygon used to construct 3D graphical objects. The two images of Figure B-3 show the use of triangles to represent a rabbit; by deforming a triangle, one can change the shape of the object.

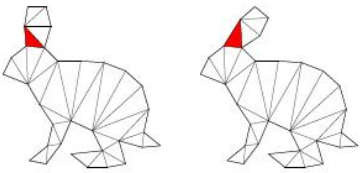


Figure B-3 Changing a Rabbit's Shape by Deforming a Polygon

There exist a lot of geometric modeling software packages that construct a model from some object, which can be a surface or a solid; it tries to capture the true shape of the object in a polygonal mesh. By using a sufficient number of polygons, a mesh can approximate the underlying surface to any desired degree of accuracy. For example, 3D Max is one of such 3D modeling software packages that run on Windows. Blender (*http://www.blender.org/*) is a popular free open-source 3D creation suite that lets users create sophisticated 3D graphical objects and do animation, and is available in all major operating systems, including Linux, Windows, and Mac OS/X. Typically, an artist creates an object using one of those packages and save it as a mesh in a file. A programmer writes programs in a computer language such as C/C++ or java to parse the polygons and manipulate the object. Artists may even post their graphical objects on some 3D graphics sites such as 3D Cafe (*http://www.3dcafe.com/*) for sale or for sharing. Specifically, **MakeHuman** (*http://www.makehuman.org/*) is an open-source tool for making 3D characters. Using MakeHuman, a photorealistic character can be modeled in less than 2 minutes and

the character can be saved as a polygon mesh. MakeHuman is released under an Open Source Licence (GPL3.0) , and is available for Windows, Mac OS X and Linux. Xface is an MPEG4-based open-source toolkit for 3D facial animation. .

Figure B-4 shows two more synthetic images for video compression taken from the web site *http://coven.lancs.ac.uk/mpeg4* that does Synthetic-Natural Hybrid Coding (SNHC) conformed to the MPEG-4 standard.

Figure B-4a Animating Sign Language

Figure B-4b Animating Business Meeting

B.2 MPEG-4 Facial Animation

To accomplish the representation of synthetic visual objects, MPEG-4 had to choose a scene description language to describe the scene structure. MPEG-4 selected Virtual Reality Modeling Language (VRML) standard as the basis with some additional new nodes to form the scene description language. Rather than constructing an object using purely triangles, the standard composes human face or body and other generic objects using a variety of geometric primitives such as cones, rectangles, triangles, and spheres. This makes the task of creating an object easier, and the object may look more realistic with the same number of polygons. To make the processing of the geometric primitives more efficient, the standard uses an indexed face set to define vertices and surface patches. It also uses nodes such as Transform to define rotation, scale, or translation and uses IndexedFaceSet nodes to describe the 3-D shape of an object. There are three problems that one has to address in animating human faces: a head needs to be specified; facial expressions have to be animated in real-time; animation and speech need to be synchronized.

MPEG-4 Facial Animation (FA) specifies the procedures to create a talking agent by standardizing various necessary parameters. The procedures of creating a talking agent consist of two phases. Phase one specifies the feature points on a static 3D model that defines the regions of deformation on the face. Phase two involves generation and interpolation of parameters that are used to modify the feature points to produce the actual animation. The two phases are cleanly separated from each other so that application developers can focus on their field of interest.

Face Definition Parameters (FDPs) and Face Animation Parameters (FAPs) are used to define head models and primitive facial expressions respectively. FDPs define the shape and texture of the model and FAPs are used to animate the face. FAPs are based on the

study of minimal perceptible actions (MPA) and are closely related to muscle actions, including movements of lips, jaw, cheek and eyebrows. They make up a complete set of basic facial actions that represent the most natural facial expressions. Exaggerated parameter values may be used for Cartoon-like characters. Figure B-5 shows some common facial expressions.

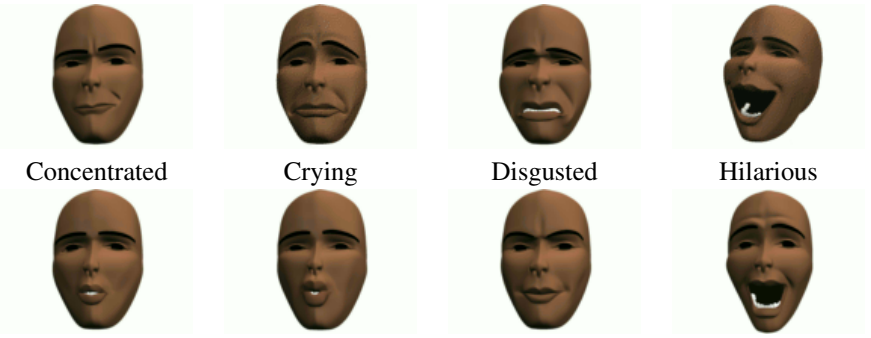


Figure B-5 Common Facial Expressions

The FAP values are usually defined and normalized in face animation parameter units (FAPUs) so that interpolations of the FAPs on any facial model are made in a consistent way. FAPUs measure spatial distances of facial expressions from its neutral state and are defined in terms of the fractions of distances between the marked key features. MPEG-4 defines a generic face model in its neutral state along with some feature points as shown in Figure B-6. The neutral state of a face model may be defined by the following features:

1. gaze is along the Z-axis,
2. all face muscles are relaxed,
3. eyelids are tangent to the iris,
4. the pupil is one third of the diameter of the iris,
5. lips are in contact,
6. the mouth is closed and the upper teeth touch the lower ones,
7. the tongue is flat, horizontal with the tip of tongue touching the boundary between upper and lower teeth,
8. FAPUs (Face Animation Parameter Units) are defined as fractions of distances between key facial features in the neutral state as shown in the following table:

Iris diameter	IRISD = IRISD0 / 1024
Eye Separation	ES = ES0 / 1024
Eye-nose Separation	ENS = ENS0 / 1024
Mouth-nose Separation	MNS = MNS0 / 1024
Mouth width	MW = MW0 / 1024
Angle unit (AU)	10^{-5} rad

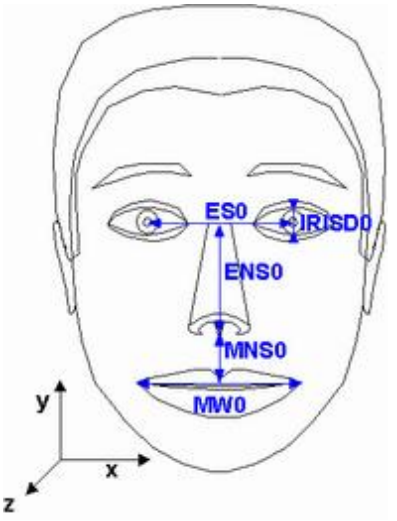


Figure B-6 A Face Model in Neutral State

For creating a standard conforming face, MPEG-4 specifies 84 feature points (FPs) on the neutral face. Feature points are arranged in groups such as cheeks, eyes and mouth. Applications need to define the locations of these feature points in order to conform to the standard. The feature points provide spatial references for defining FAPs as well as calibration between models when switched from one player to another. Figure B-7 shows the set of FPs which are used to provide spatial reference for defining FAPs. The 68 FAPs are classified into 10 groups as shown in the following table:

Table B-1 FAP Groups

Group	Number of FAPs
1. visemes and expressions	2
2: jaw, chin, inner lowerlip, cornerlips, midlip	16
3. eyeballs, pupils, eyelids	12
4. eyebrow	8
5. cheeks	4
6. tongue	5
7. head rotation	3
8. outer lip positions	10
9. nose	4
10. ears	4
Total	68

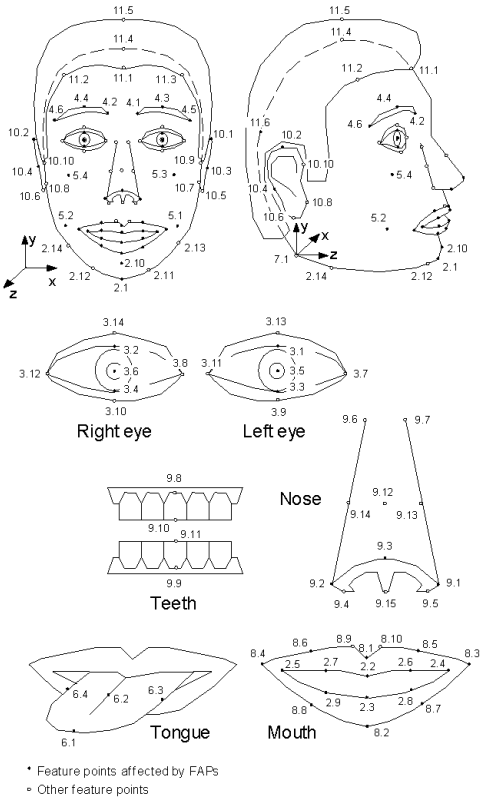


Figure B-7 MPEG-4 Feature Points

B.3 Computing Face Mesh Vertices

We use FAP values to animate a facial model, creating desired facial expressions. MPEG-4 further divides the FAPs into two subgroups. The first subgroup consists of FAPs that control simple motion of human face such as rotation, translation and scaling. The second subgroup FAPs are used for animating more complex motions that do not have any regular order such as frowning, blinking, and mouth-opening.

The first subgroup FAP values are fairly easy to process. For example, FAP23 is used to animate horizontal orientation of left eyeball. Suppose we have the following parameters,

AU (Angle Unit)	=	10^{-5} rad
Rotation Axis	=	(0, -1, 0)
Rotation factor θ	=	1
Value of FAP23	=	10000

then the left eyeball needs to be rotated by an angle α given by,

$$\alpha = 10^{-5} \times 10000 \times 1 = 0.1\ radian$$

The mesh vertex oordinates are more difficult to obtain from the second subgroup of FAPs. We have to perform a piecewise linear interpolation to obtain the new vertex coordinates of the mesh in the affected region. Figure B-8 shows two phases of a left eye blink

along with the neutral phase. The eyelid motion is controlled by FAP19. In the blinking animation, the eyelid movement is along an acred trajectory but we can using 2D coordinates to specify the trajectory as shown in the figure.

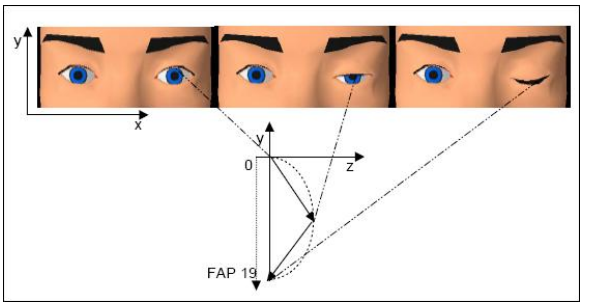
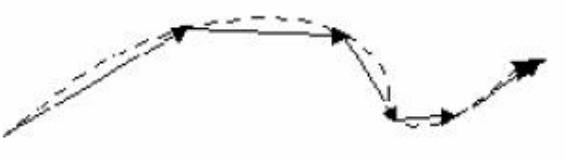


Figure B-8 Two Phases of Movement of Upper Left Eyelid

In general, we can compute the displacements of mesh vertices using piecewise linear interpolation. We approximate the motion trajectory of each mesh vertex as a piecewise linear one as shown in the figure below:



Suppose P_m is the position of vertex m when the face is in neutral state (FAP = 0) and $D_{m,k}$ is the 3D displacement that defines the piecewise linear function in the kth interval as shown in Figure B-9. If P'_m is the new position of the same vertex after animation with the given FAP value, we can compute

P'_m according to the following algorithm which is slightly different from that of MPEG-4, which requires 0 to be on an interval boundary all the time.

1. Assume that the range of FAP is divided into max intervals:

$$[I_0, I_1], [I_1, I_2], [I_2, I_3], ..., [I_{max-1}, I_{max}]$$

where

$$I_0 = -\infty,\ I_{max} = +\infty$$

2. Assume that the received FAP is in the jth interval, $[I_j, I_{j+1}]$, and 0 is in the kth interval, $[I_k, I_{k+1}]$, with $0 \leq j, k < max$. (See Figure B-9.)

3. If $j > k$, we compute the new position P'_m of the mth vertex by:

$$\begin{aligned} P'_m = \quad & P_m + FAPU \times [(I_{k+1} - 0) \times D_{m,k} + (I_{k+2} - I_{k+1}) \times D_{m,k+1} \\ & + ... + (I_j - I_{j-1}) \times D_{m.j-1} + (FAP - I_j) \times D_{m,j}] \end{aligned} \tag{B.1}$$

4. If $j < k$, we compute P'_m by:

$$\begin{aligned} P'_m = \quad & P_m + FAPU \times [(I_{j+1} - FAP) \times D_{m,j} + (I_{j+2} - I_{j+1}) \times D_{m,j+1} \\ & + ... + (I_k - I_{k-1}) \times D_{m.k-1} + (0 - I_k) \times D_{m,k}] \end{aligned} \tag{B.2}$$

5. If $j = k$, we compute P'_m by:

$$P'_m = P_m + FAPU \times FAP \times D_{m,k} \tag{B.3}$$

6. If the range of FAP contains only one interval, the motion is strictly linear, and we compute P'_m by:

$$P'_m = P_m + FAPU \times FAP \times D_{m,0} \tag{B.4}$$

For example, suppose the FAP range is divided into three intervals:

$$[-\infty, 0], [0, 500], [500, +\infty].$$

The coordinates (x, y, z) of the displacements of vertex m controlled by the FAPs in these intervals are:

$$\begin{pmatrix} 1 \\ 0 \\ 2 \end{pmatrix}, \qquad \begin{pmatrix} 0.8 \\ 0 \\ 0 \end{pmatrix}, \qquad \begin{pmatrix} 1.5 \\ 0 \\ 4 \end{pmatrix}$$

respectively. The coordinates of vertex m in neutral expression is P_m. Suppose the received FAP value is 600 and the corresponding FAPU is Mouth Width, MW = 0.1. Since this FAP value is in the third interval [500, $+\infty$] and 0 is in the second interval [0, 500], we have the situation $j > k$. Thus we apply (B.1) to calculate the new position P'_m of vertex m:

$$P'_m = P_m + 0.1 \times [(500 - 0) \times \begin{pmatrix} 0.8 \\ 0 \\ 0 \end{pmatrix} + (600 - 500) \times \begin{pmatrix} 1.5 \\ 0 \\ 4 \end{pmatrix}] = P_m + \begin{pmatrix} 55 \\ 0 \\ 40 \end{pmatrix}$$

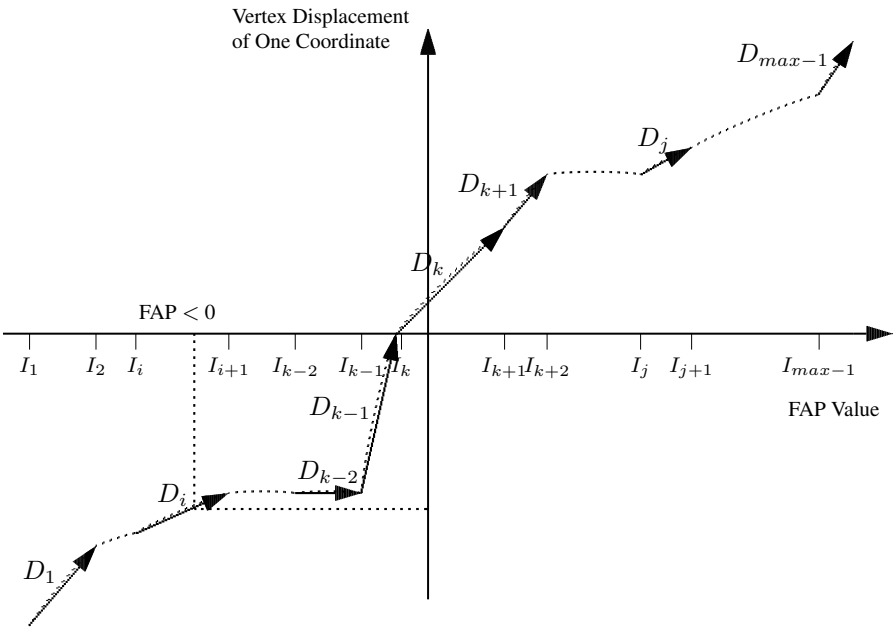
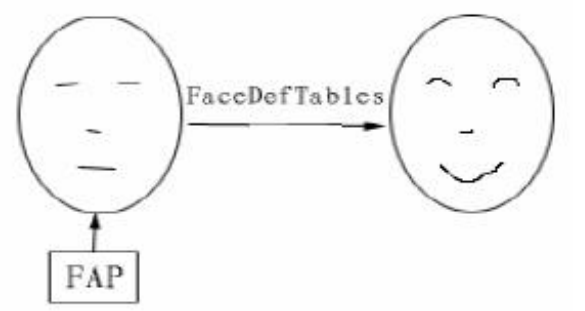


Figure B-9. Piecewise Linear Interpolation of FAP Values

To speed up the animation process, we may save the relation between FAP intervals and 3D displacements in the so called *FaceDefTables*. When we get the value of an FAP, we

need to look up the *FaceDefTables* to get information about the control region of the FAP and the three dimensional displacements of vertices within the control region to convert the FAP into facial animation as shown in the figure below:



B.4 Keyframing

To animate realistic facial expressions, one can first calculate the animation parameters for key frames from photographs. A key frame in animation and film making is an image that defines the starting and ending points of any smooth transition. **Keyframing** is the process of creating animated motion by specifying objects at key frames and then interpolating the motion in intermediate frames. For traditional animation of movies, the key frames, also known as keys, are drawn by a senior artist. Other artists, *inbetweeners* and *inkers*, draw the intermediate frames and fill in the complete detailed drawings. The key frames are drawn for critical poses or when there is a sharp change in motion. At the beginning, the key frames could be very brief without fine details. The inbetweener would draw the intermediate poses to ensure that the motion appear fluid and natural.

Keyframing is particularly important in the animation of a full-length movie, which plays 24 frames per second. A 90-minute movie consists of 129,600 frames, which are way too many for a single artist to handle. On the other hand, if many artists draw different portions of the movie, the style or even appearance could be inconsistent. The movie has much better look if the production company employs a few senior animators to draw the key frames and a larger number of inbetweeners and linkers to draw the frames in between. A single senior animator can draw all the key frames of a particular character in a movie, producing more consistency of style.

In a similar spirit, computer animation uses interpolation to do keyframing. We specify the crucial parameters such as positions, orientations, and shapes of objects in the scene at key frames. Then, we obtain the parameters as smooth functions of time by using interpolating curves. This often can be done fully automatically but manual editing may be needed at some stages.

B.5 Extracting FAPs From Video

We have discussed how to use FAPs to animate facial expressions. This actually is a relatively easy part. When the animation parameters are given, one can basically use any appropriate model to perform animation. The more difficult problem is how to extract the

FAPs from a given video. The extraction of facial parameters from video is not a completely solved problem. The process involves face detection, identification, recognition and tracking which are still active research topics in the academic and the industry. Automatic and accurate location of facial features is always difficult. The variety of human faces, expressions, facial hair, glasses, poses, and lighting contribute to the complexity of the problem.

A quick and simple way to find a human face in an image is to search for some common characteristics of human faces, such as color and shape. Some people use Delaunay triangulation and Voronoi diagrams to locate facial features of humans. However, this method is usually not very robust.

More sophisticated methods in general involve some statistical techniques and deformable models. A human is a deformable object and the tracking and recognition of it is usually tackled by making simplified assumptions concerning the motion or imposing constraints on it. Very often, the first step towards human tracking is to segment human figures from the background. A popular and relatively simple method to extract and track a deformable object in an image is the Active Contour Model (also called "snake"), which finds the contour of an object by balancing the effects of several energy terms. Variations of the Active Contour Model also exist. The gradient vector flow (GVF) snake is an improved model that includes the gradient vector flow, a new non-irrotational external force field. Another approach for tracking a deformable object is to employ deformable templates to automatically detect the objects. In general, retrieval by shape requires object detection and segmentation. Some researchers had used model-based region-grouping techniques to detect and retrieve deformable objects and found that using this method along with perceptually-motivated splitting strategy yields good image segmentation results of deformable shapes. In this section, we present a practical and popular technique for extracting FAPs from a video, the active appearance model (AAM).

B.5.1 Active Appearance Model (AAM)

An active appearance model (AAM) considers a face as a pattern in an image and makes use of a statistical model to match its appearance and shape with a pattern in the image. The approach is widely used for matching and tracking faces and for medical image interpolation.

AAM is an extension of the active shape model (ASM), which is a statistical model of the shape of objects that iteratively deform to fit to an example of the object in a new image. The shapes are constrained by the PDM (point distribution model) Statistical Shape Model to vary only in ways seen in a training set of labeled examples.

The shape of an object can be represented by a mesh of polygons or represented by a set of points (controlled by the shape model). Typically, it works by alternating the following steps:

1. Look in the image around each point for a better position for that point.
2. Update the model parameters to best match these new found positions.

To locate a better position for each point one may need to look for strong edges, or a match to a statistical model of what is expected at the point.

Usually the points that represent a shape are referred to as *landmarks*. In general, a *landmark* is a distinguishable point present in most of the images under consideration and

people use landmarks to locate features of an image. In our case, we locate facial features by locating landmarks. Figure B-10 shows an image with correctly positioned landmarks.

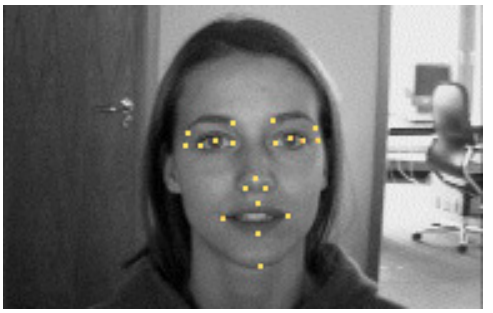


Figure B-10 A face with correctly positioned landmarks

A set of landmarks forms a shape. Therefore, a shape s consists of a set of points. We can express s as an n-tuple:

$$s = \begin{pmatrix} p_1 \\ p_2 \\ . \\ . \\ . \\ p_n \end{pmatrix} \tag{B.5}$$

where

$$p_i = \begin{pmatrix} x_i \\ y_i \\ z_i \end{pmatrix} \tag{B.6}$$

is a point with three coordinates if we consider 3D space. The z_i component of (B.6) will be dropped if we consider 2D space. In general, the points of a shape are vertices of a mesh composed of triangles.

One can align one shape to another with an affine transformation (translation, scaling, or rotation) that minimizes the average Euclidean distance between shape points. The mean shape is the mean of the aligned training shapes, which are manually landmarked faces. (Note that the average of points is a linear affine combination of points and thus the average is also a valid point.) In general, a shape s is typically controlled by adding a linear combination of shape/deformation modes to the average shape $\overline{s}$:

$$s = \overline{s} + \Phi b \tag{B.7}$$

where b is a set of vectors consisting of deformation parameters and Φ is a matrix whose columns contain the deformation modes. A deformation is a displacement of a point and can be regarded as a vector. The operation Φb gives us another set of vectors. Therefore, in (B.7) we add a set of points to a set of vectors and the operation is legitimate; the result is another set of points as the sum of a point and a vector yields another point.

We can generate various shapes with Equation (B.7) by varying the deformation parameters in b. By keeping the elements of b within limits (determined during model building) we ensure that the generated face shapes are lifelike. Conversely, given a suggested shape s, we can calculate the parameter b that allows Equation (B.7) to best approximate s with

a model shape s'. One can use an iterative algorithm to find b and T that minimize a 'distance' D described by

$$D = ||s, T(\overline{s} + \Phi b)|| \tag{B.8}$$

where T is an affine transformation that maps the model space into the image space.

ASM is relatively fast but it is too simplistic, not robust when new images are introduced. It may not converge to a good solution. Another disadvantage of ASM is that it only uses shape constraints and does not take advantage of all the available information like the texture of the target object.

It turns out that an equation similar to (B.7) can be also used to describe the texture of objects based on statistical models. A texture t of an object is also a set of points at various locations of the objects:

$$t = \overline{t} + \sigma w \tag{B.9}$$

where $\overline{t}$ is the average texture, σ describes texture modes and w is a set of texture parameters.

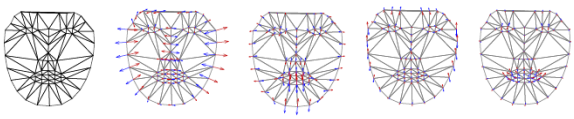
Active appearance models (AAMs), also known as "smart snakes" as they conform to some explicit shape constraints like what an active contour model (snake) does, combine shape and texture into a single statistical model. We can express an AAM as

$$\begin{aligned} s &= \overline{s} + \phi \mathbf{v} = \overline{s} + \sum_{j=1}^{n} \phi_{ij} v_j \\ t &= \overline{t} + \sigma \mathbf{v} = \overline{t} + \sum_{j=1}^{n} \sigma_{ij} v_j \end{aligned} \tag{B.10}$$

That is, we use the same displacement vectors to control both shape and texture. In (B.10), each v_i is a vector and the coefficients ϕ_i and σ_i are shape and texture parameters respectively. Note that

$$v_i = \begin{pmatrix} v_{ix} \\ v_{iy} \\ v_{iz} \end{pmatrix}$$

AAMs are normally computed from training data. The standard approach is to apply Principal Component Analysis (PCA) to the training meshes. The mean shape $\overline{s}$ is usually referred to as the base shape. Figure B-11 shows an example of AAM, where $\overline{s}$ is the average shape and v_i's are shape vectors.



$\overline{s}$ v_1 v_2 v_3 v_4

Figure B-11 An Example of AAM. $\overline{s}$ is the average shape. v_i's are shape vectors.

B.5.2 An AAM Search Algorithm

Face modeling has been the most frequent application of AAMs. Typically an AAM is first fit to an image of a face. That is, we search for model parameters that maximize the "match" between the model instance and the input image. The model parameters are

then used in whatever the application is. Fitting an AAM to an image is a non-linear optimization problem. The usual approach is to iteratively solve for incremental additive updates to the parameters (the shape and appearance coefficients.) We briefly describe an AAM search algorithm here that is based on such an iterative update.

For an input image I and a model parameter vector $\mathbf{v}$, we can map the image onto the model and reshape the model to the standard shape to create a normalized image J:

$$J = J(I, s(\mathbf{v})) \tag{B.11}$$

For simplicity, we assume that the input image I is fixed. Therefore, the normalized image J is a function of $\mathbf{v}$ only. The residual image R is given by

$$R(\mathbf{v}) = J(\mathbf{v}) - s(\mathbf{v}) \tag{B.12}$$

We want to find $\mathbf{v}$ so that the error measure

$$E(\mathbf{v}) = ||R(\mathbf{v})||^2 \tag{B.13}$$

is minimized. Suppose we roughly know that the optimal $\mathbf{v}$ is near $\mathbf{v_0}$. Then the optimization process can be approximated by finding $\delta\mathbf{v}$ so that $E(\mathbf{v_0} + \delta\mathbf{v})$ is optimized. We can make a Taylor expansion of $R(\mathbf{v_0} + \delta\mathbf{v})$ and simplify the expression by retaining only the first two terms:

$$R(\mathbf{v_0} + \delta\mathbf{v}) \approx R(\mathbf{v_0}) + D\delta\mathbf{v} \tag{B.14}$$

where

$$D = \frac{\partial R(\mathbf{v})}{\partial \mathbf{v}}$$

is evaluated at $\mathbf{v} = \mathbf{v_0}$. Thus, our optimization is reduced to minimizing

$$E(\mathbf{v_0} + \delta\mathbf{v}) \approx ||R(\mathbf{v_0}) + D\delta\mathbf{v}||^2 \tag{B.15}$$

The least square solution to Equation (B.15) is

$$\delta\mathbf{v} = -(D^T D)^{-1} D^T R(\mathbf{v_0}) \tag{B.16}$$

We can use Equation (B.16) to update $\mathbf{v}$ in the search space; we use the $\delta\mathbf{v}$ to compute a new vector $\mathbf{v}$ and a new error measure:

$$\begin{aligned} \mathbf{v}' &= \mathbf{v_0} + \delta\mathbf{v} \\ E' &= E(\mathbf{v}') \end{aligned} \tag{B.17}$$

If $E' < E$, we update $\mathbf{v}$ accordingly ($\mathbf{v}' \rightarrow \mathbf{v_0}$) and repeat the steps until convergence occurs. If $E' > E$, we do not perform the update but try smaller update steps. If the smaller steps still do not improve the error measure, we assume that convergence has reached.

We can estimate the gradient matrix D from a set of training data. For example, the ith row of D can be estimated as:

$$D_i = \sum_k [R(\mathbf{v} + \delta\mathbf{v}_{ik}) - R(\mathbf{v})] \tag{4.18}$$

where $\delta\mathbf{v}_{ik}$ is a vector that perturbs $\mathbf{v}$ in the ith component to the amount of $k \times c$ for some suitable constant c. We can then compute the update matrix U as the negative pseudoinverse of D:

$$U = -D^* = -(D^T D)^{-1} D^T \tag{B.19}$$

We can apply a similar procedure to the texture of Equation (B.10).

Some tools for studying AAM can be downloaded from the web site of Professor Tim Cootes (*http://personalpages.manchester.ac.uk/staff/timothy.f.cootes/*).

B.6 Conclusions

The application of graphics techniques to video compression has become an interested research topics in recent years. If your interest is mainly on utilizing existing video compression libraries, you may use open-source video codecs for your applications. One can make use of the libraries to easily integrate videos in a graphics or video game application. The use of these open-source video compression libraries will tremendously shorten your software development cycle. Because of the huge advancement and demand of Internet and multi-media applications, video technologies and computer graphics are blending together more each year. The technologies will be even more important in the coming decades. Their applications are only limited by your imagination.

Other books by the same author

Windows Fan, Linux Fan

by *Fore June*

Windws Fan, Linux Fan describes a true story about a spiritual battle between a Linux fan and a Windows fan. You can learn from the successful fan to become a successful Internet Service Provider (ISP) and create your own wealth. See *http://www.forejune.com/*

Second Edition, 2002.
ISBN: 0-595-26355-0 Price: $6.86

An Introduction to Digital Video Data Compression in Java

by *Fore June*

The book describes the the principles of digital video data compression techniques and its implementations in java. Topics covered include RBG-YCbCr conversion, macroblocks, DCT and IDCT, integer arithmetic, quantization, reorder, run-level encoding, entropy encoding, motion estimation, motion compensation and hybrid coding. See

http://www.forejune.com/

January 2011

ISBN-10: 1456570870

ISBN-13: 978-1456570873

An Introduction to Video Compression in C/C++

by *Fore June*

The book describes the the principles of digital video data compression techniques and its implementations in C/C++. Topics covered include RBG-YCbCr conversion, macroblocks, DCT and IDCT, integer arithmetic, quantization, reorder, run-level encoding, entropy encoding, motion estimation, motion compensation and hybrid coding.

January 2010
ISBN: 9781451522273