POLITECNICO DI TORINO

SCUOLA DI DOTTORATO

Dottorato in Ing. Elettronica e delle Communicazioni – XXVI ciclo

Tesi di Dottorato

3D Graphics Reconstruction, Compression and Animation

Akhlaque AHMAD

Tutore
Prof. Gabriella OLMO

Coordinatore del corso di dottorato
Prof. Ivo Montrosset

Febbraio 2014

Acknowledgements

I wish to express my gratitudes to people at STMicroelectronics for supporting this work, and specially to Massimiliano Barrone whose endless help and unbounded support, made me able to write this dissertation. His supervision, guidance and broad vision always kept me on the right track. I also pay my sincere gratitudes to Prof. Gabriella OLMO, Tea ANSELMO, Daniele ALFONSO and Emanuele QUACCHIO who always motivated me to explore the new dimensions in the work.

This work has been supported by the European Community's Seventh Framework Programme (FP7/2007-2013) under grant agreement no. ICT-2011-7-287723 (REVERIE project).

Summary

This work covers entire pipeline of a 3D immersive system, comprising acquistion and reconstruction of 3D objects, transforming the 3D objects into popular 3D format, compression of data with MPEG-4 compliant encoders and acquisition of animation motion data.

For the acquisition part we created 3D human face object with the help of depth-field camera, in particular we used Microsoft Kinect. We process the raw data given by Kinect and transform it into a mesh, and texture it with photometric data. The reconstructed objects are quiet large in size and need to be compressed for an efficient network transmission. We encoded the objects using MPEG-4 encoder, and measured the performance of scalable mesh encoding techniques.

For an interactive immersive application, motion data of player must be captured, which is transmitted to remote client for playing the animation on the virtual character of player. For this purpose MPEG-4 standard defines a Bone Based Animation to acquire and compress the motion data. We acquired the motion data of a player using a novel algorithm which is compliant to bone based animation. Our proposed approach of extracting motion data is computationally efficient. We tested it with 3D graphics player developed at STMicroelectronics.

Contents

Chapter 1

Introduction

Computer graphics are being used in many applications, such as computer aided design, 3D games, immersive environments, virtualization of medical, cultural heritage and other 3D environments. Photorealistic 3D contents creation became much simple with the availability of depth field sensors. We can reconstruct a 3D object while processing the data given by depth sensors e.g. Microsoft Kinect. There is an increasing demand for network-based distribution of 3D graphics coming from online media sharing services and distributed applications. The amount of 3D data in such applications are usually very large, and processing is too complex, that lead to development of many authoring tools specialised in their particular tasks. The interoperability of 3D graphics contents in these authoring tools leads to development of exchange formats such that eXtensible 3D (X3D) [4] and COLLaborative Design Activity (COLLADA) [3]. These tools represent the data in eXtensibile Markup Language (XML) [14] format, in which the redundancy and verbosity imposes the need of compression to reduce the time required to transmit 3D models over digital communication channels. Generic XML compression techniques reduce the size of data by order of 10 by exploiting only the data structure redundancy [6]. Applying a lossy compression by exploiting spatial and temporal correlation can reduce the data size by the order of 40 [15]. In this work we applied compression techniques standardized by MPEG, and we achieved compression efficiency more than 80 %.

For an interactive immersive application, motion data of player must be captured, which is transmitted to remote client for playing the animation on the virtual character of player. For this purpose MPEG-4 standard defines a Bone Based Animation to acquire and compress the motion data of player.

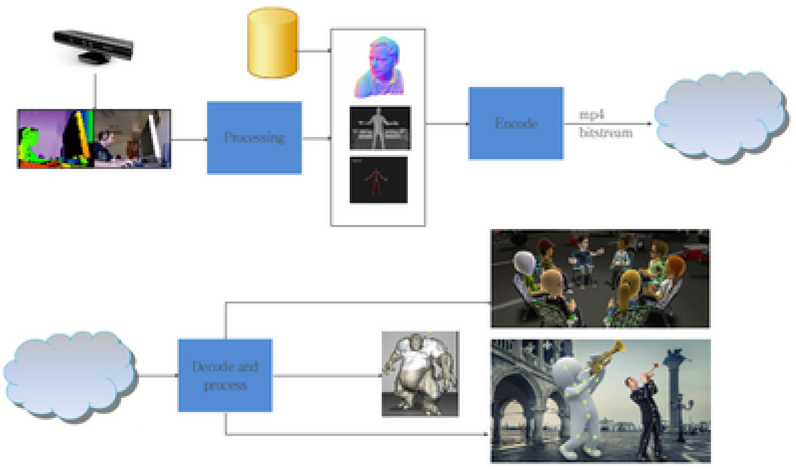


Figure 1.1. A big picture of system

A potential fully developed system is depicted in Fig. 1.1, where a user interacts with the system capable of capturing 3D data. This data is processed and depending upon the target application a 3D content is created, that is transmitted to the network or stored in a database. This content is compressed, and sent to the network, where on a remote location it is decompressed and processed further for required task. The applications of this work can be in gaming, medical, training and other commercial activities. For example the users can participate in a virtual meeting room with their avatar presentation, or they can play mutually a virtual reality based game. In medical field, such applications can be targeted where patients require training for their recommended exercises while sitting in their home.

This work covers entire pipeline of a 3D immersive system, comprising acquistion and reconstruction of 3D objects, transforming the objects into popular 3D format, compression of data with MPEG-4 compliant encoders and acquisition of animation motion data.

The organization of this thesis is as follows: Chapter 2 explains some state of art acqusition systems, comprising silhoutte based, image based and hybrid camera based reconstruction systems. Then we propoed our strategy for the reconstruction of human face, as it is desired by video conferencing applications with synthetic avatars.

Chapter 3 explains various techniques for mesh compressions. We explained the geomtry and connectivity compression methods. In particular MPEG-4 activities on graphics compression are discussed. The compression technique adopted in MPEG-4 part 25, scalable complexity mesh coding, is used to experiment graphics comperssion of large data set of scanned human body, and encoding efficiency are reported.

In Chapter 4, we discussed the bone based animation method of MPEG-4 for animation of avatars. We report our skeleton extraction method that is compliant with bone based animation. Then we explained motion data representation format and how we configured motion data with a grphics player. Chapter 5 concludes the work and proposes some possible future extension for this work.

Chapter 2

3D Graphics Acquisition

Most existing video conferencing systems make some attempt to humanize remote interaction, but few are able to provide the desired immersive component of actual face-to-face communication. These systems, which rely on two dimensional video streams between remote users, lack to provide the desired immersive component for a number of reasons, e.g., not allowing users to establish eye contact, not placing all users inside the same environment, or not allowing users to jointly interact with synthetic objects. Few attempts [1, 2] have been made to create a more immersive experience using large displays, eye contact through multi-camera capturing systems, and matching environments (e.g., tables, chairs) between the remote locations that create the illusion of continuity of the physical space into the screen.

In contrast to such systems, an immersive experience is one that generates a full body real-time 3D reconstruction that realistically represents a user's life sized appearance. It completely models the dynamics of movement such as facial expressions, gesture, postures, and provides the mechanism so that remote users can jointly interact with synthetic objects.

2.1 Teleimmersive Environments

Teleimmersive environments are setup to capture the 3D scene. Such environments could be a dedicated room with multiple cameras surrounding the entire room or a PC with specific capturing apparatus that is capable to acquire 3D scene. Several attempts have been made to develop systems for 3D reconstruction of human body to present in a virtual environment. Acquisition of 3D objects are performed with different reconstruction methods, which are given as below.

Existing teleimmersive systems achieve various levels of presence, depending on the number and layout of cameras and the 3D modeling algorithm used. Some systems offer a free viewpoint on the observed user [22] , enabling the visual presence of the user. Other systems compute a partial 3D model based on a depth map [33, 21]. Since they capture the scene from a specific direction only, they provide a partial visual and geometrical presence, for instance free viewpoint can't be achieved or thickness of modeled objects is unknown.

In the past decade, several attempts have been made to develop 3D reconstruction systems for marker-less capture of the human body for telepresence in a virtual environment. Most real-time approaches fall into one of three categories depending upon their computational approach:

- **Silhouette Based Reconstruction:** Objects are captured with multiple cameras surrounding the complete environment
- **Image Based Reconstruction:** Stereoscopic images are processed to model the objects
- **Hybrid Camera Based Systems:** Uses Time-of-Flight cameras to compute the depth values of objects

3D model of objects are reconstructed using any of the above methods, which is then transmitted to remote locations through TCP/IP or any other preferred way in realtime. The objects are recovered on the remote locations and placed in a virtual shared environment among all the users of the system. The movements of objects are rendered in the virtual environment so that feel of presence could be achieved. In the following sections each of reconstruction methods are explained outlining an implemented system based on that technique.

2.2 Silhouette Based Reconstruction

To improve the sense of presence and realism, models with both photometric and geometric information are considered. They yield more realistic representations that include user appearances, motions and even sometimes facial expressions. In addition to appearance, through photometric information, they can provide a hierarchy of geometric representations from 2D to 3D, including 2D and depth representation, multiple views, and full 3D geometry. 2D and depth representations are viewpoint dependent and though they enable 3D visualization and, to some extent, freeviewpoint visualization.

Multiple view representations, that is, views from several viewpoints, overcome some of the limitations of 2D and depth representations. In particular, they increase the freeviewpoint capability when used with view interpolation techniques. However, interpolated view quality rapidly decreases when new viewpoints distant from the original viewpoints are considered. And similarly to 2D and depth representations, only limited interactions can be expected. In contrast, full 3D geometry descriptions allow unconstrained free viewpoints and interactions as they carry more information.

A system developed at INRIA [23] is based on this technique where multiple cameras are set to surround the scene which constructs a visual hull of the objects of the interest. The visual hull is textured with the photometric data obtained from the cameras to fully model a 3D representation. The number of cameras depends on the size of scene and the visual quality we want to acquire. The location of cameras can be chosen as the required preference to emphasize any particular side of the scene.

2.2.1 Synchronization

All the cameras used in the setup need to be synchronized so that all images coming from different cameras are coherent. Synchronization can be achieved directly from silhouettes using inconsistency over several viewpoints. But a better hardware solution is implemented in this project.

2.2.2 Calibration

Spatial organization of cameras need to be determined when dealing with multiple cameras in order to perform geometric computations. In practice we need to determine the position and orientation of each camera in the scene as well as its intrinsic characteristics such as the focal length. This is done through a calibration process that computes the function giving the relationship between real 3D points and 2D-image points for each camera.

2.2.3 Background Subtraction

Regions/Objects of interest are to be extracted from the scene by eliminating the background. High quality of background subtraction is achieved by using a dedicated priory known background.

2.2.4 3D Modeling

To model a 3D object shape from silhouette method is implemented that builds a visual hull of the objects. This visual hull is then textured with the photometric data to form a complete 3D object which can be represented in virtual environment.

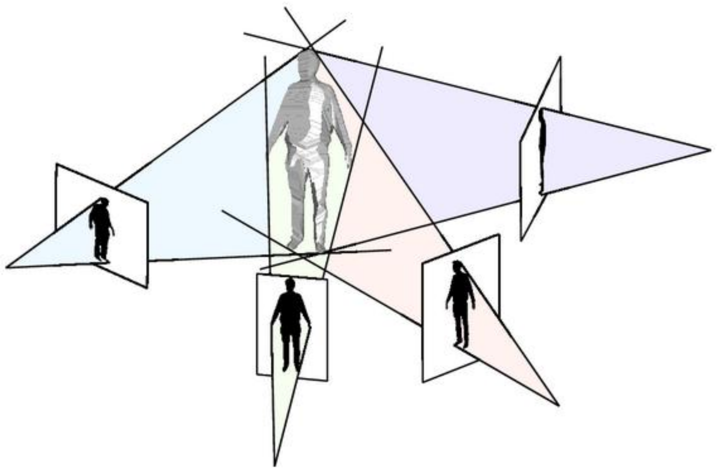


Figure 2.1. A visual hull of a person with 4 views

2.2.5 Visual Hull

The visual hull is a well-studied geometric shape which is obtained from scene object's silhouettes observed in n views. Geometrically, the visual hull is the intersection of the viewing cones, the generalized cones whose apices are the cameras' projective centers and whose cross sections coincide with the scene silhouettes as show in the Fig. 2.1. Although visual hull can't model concavities in the object but it can give a good approximation of the object shape.

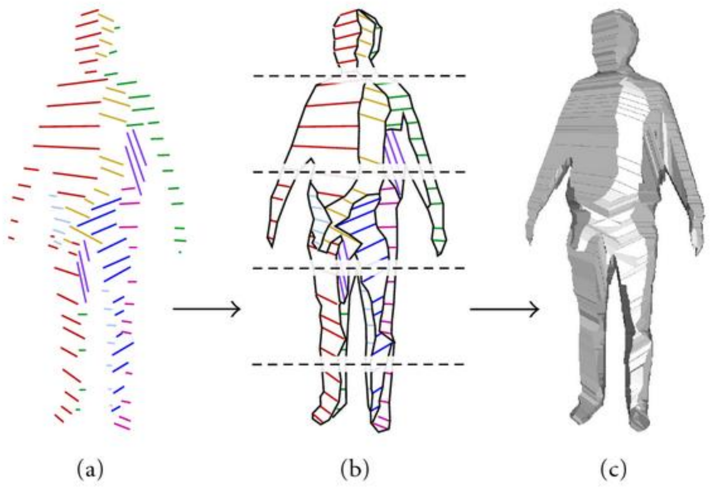


Figure 2.2. The three main steps of visual hull construction, (a) viewing edge computation, (b) mesh connectivity, and (c) face generation.

Three steps are performed to create a visual hull of the object. First, a particular subset of the polyhedron edges is computed, the viewing edges, which is the projection on a viewing line from all other views. Second, all other edges of the polyhedron mesh are recovered by a recursive series of geometric deductions. The positions of vertices not yet computed are gradually inferred from those already obtained, using the viewing edges as an initial set. Third, the mesh is consistently traversed to identify the faces of the polyhedron. The steps are shown in Fig. 2.2

2.3 Image Based Reconstruction

Depth map can be calculated by stereoscopic processing of two images taken few distance apart. A stereo video signal captured by two input cameras is first preprocessed. This includes possible image rectification for row-wise left and right view alignment, as well as color and contrast correction due to possible differences between the input cameras. The 3D format is called Conventional Stereo Video (CSV) for left and right view. This format is encoded by multiview coding methods, such as specified in the stereo high profile of H.264/AVC, where temporal dependencies in each view, as well as interview dependencies between both views are exploited for efficient compression. Given a stereoscopic pair of left and right images (IL, IR), the disparity function $d(x, y)$ is defined as:

$$I_R(x, y) = I_L(x + d(x, y), y) \tag{2.1}$$

Traditional methods to determine disparity $d(x, y)$ either employ a local approach like normalized cross correlation or global optimization techniques. The normalized cross correlation techniques match a fixed sized window assuming that all pixels inside that window have same depth, which is quiet unreasonable, although this assumption so much reduces the processing time. Global optimization techniques begin by associating a cost to each pixel's disparity that depends not only on how well it matches to a pixel in the other image, but also how well this disparity matches to neighboring pixels' disparity. This cost function is then minimized in order to determine the disparity of each pixel. Though these optimization techniques produce more accurate results than the entirely local approaches, they are computationally expensive.

Considering the limitations of both local and global optimization techniques, a hybrid technique is applied [32] to find the disparity map using an image pair. A mesh of triangles is formed, where each triangle dictates the level of depth of each pixel in the triangle, i.e. each pixel in a triangle have approximately same depth. A coarse mesh of triangles is formed at level 1, and then each triangle is bisected if the amount of variation of depth among the pixels exceeds a user defined threshold. So as a result the regions with more varying disparities are represented with small triangles while the regions with small variations are represented with large triangles.

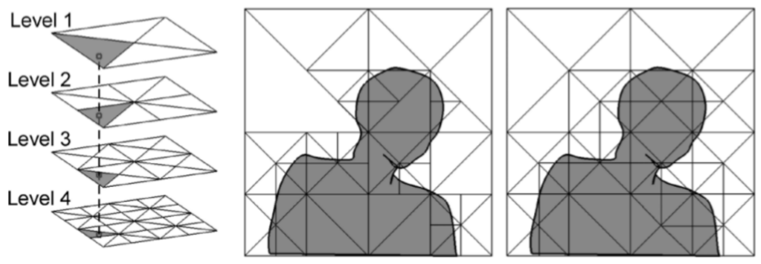


Figure 2.3. Triangle Bisections

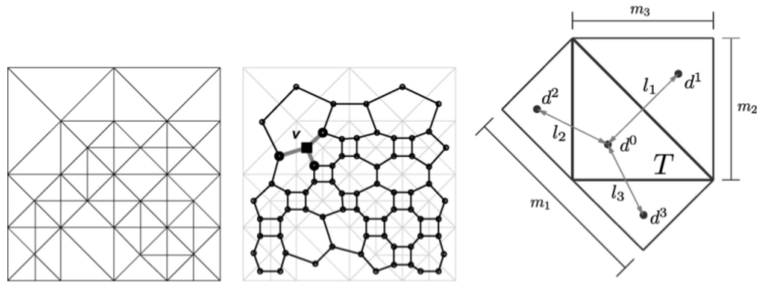


Figure 2.4. Representation of triangular mesh

2.3.1 Image Model

Before generating the triangular mesh of the image domain, a coarse mesh of right isosceles triangles is generated at level $n = 1$ as illustrated in Fig. 2.3. This coarse mesh is then refined by bisecting each triangle. The refinement procedure in a particular region is halted when the variation in each triangle is less than a user specified threshold τ. The goal is to mimic the effects of a global optimization procedure by refining initial local disparity estimates via anisotropic diffusion. To define anisotropic diffusion, the mesh must be able to share information between neighboring nodes which requires that the mesh have no nodes that are on the middle of another triangle's edge. A mesh satisfying such a property is referred to as a conforming mesh. A conforming triangular mesh is constructed by following Algorithm 1.

The procedure of bisecting the triangles is shown in Fig. 2.3. A graphical representation, of the image is then constructed by letting each triangle in the mesh correspond to a vertex and letting edges in the graph correspond to triangles that share an edge. Fig. 2.4 illustrates the construction of such a graph.

2.3.2 Disparity Calculation

To calculate disparity map, left and right images are matched. The matching score is defined as the average normalized cross correlation between the reference window centered at each of the corners of i in the right image and a window centered at the same coordinate as each of the corners of i in the left image after some horizontal translation by r. Importantly the size of the neighborhoods used during the normalized cross correlation step are dictated by the level of the triangle i. Initial disparity is assumed as the point that gives the highest cross correlation. This initial disparity is further improved by post processing techniques.

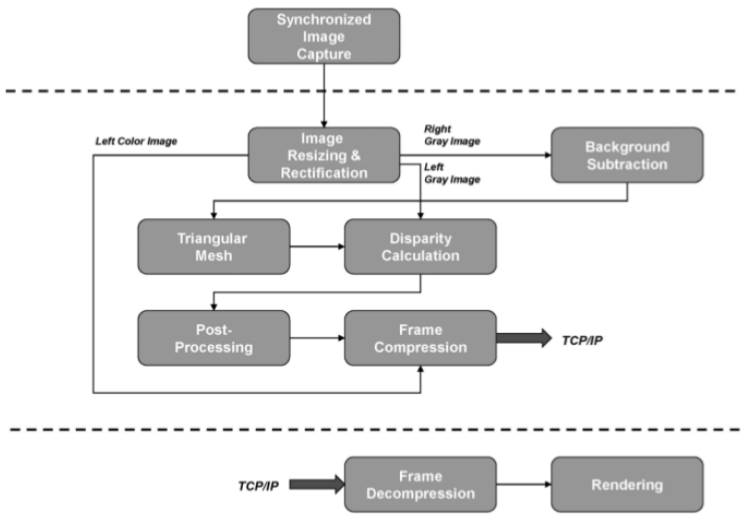


Figure 2.5. Stereo algorithm for 3D mesh generation

2.4 Hybrid Camera Based Systems

Recently, fusion methods that combine video cameras and a time-of-flight (TOF) depth camera have been introduced. Depth can also be obtained directly from a camera if special light, called Near Infrared (NIR), is used. This depth information when integrated with a photometric data produces the complete model of 3D video representation.

NIR light has a series of advantages over normal light or structured light methods. First, it is insensitive to lighting conditions, and thus the image processing part of the 3D object reconstruction might be more stable than in normal daylight or in spaces with artificial light. Second, when reflected by the object, it is possible to detect the depth directly via the intensity of every pixel. This is because the reflected NIR light intensity, if correctly captured by the image sensor, can be made proportional to the distance. While it provides very poor pixels resolution, which are up-sampled to map them with photometric data resulting in unrealistic depth information of, in particular, boundaries and edges of the objects. [17] proposed a system that uses hybrid camera system, the features of this system is outlined below.

2.4.1 Relative Camera Calibration

Since two different types of cameras are being used, so relative camera calibration information are essential to find. For this purpose, the projection matrices of all cameras are obtained and compared to each other on the basis of intrinsic characteristics, rotation matrices and translation vectors.

2.4.2 Depth Calibration

Depth information of the depth camera is very sensitive to color and motion. Even if the distance from the depth camera to the object is constant, depth information from depth camera is different depending on the environment. Usually, the depth camera system has its own depth calibration tool, but it is very poorly calibrated. A mapping curve is determined by placing the objects at different positions of the scene and checking the depth of objects from depth camera. This mapping is then used to compensate the difference between actual depths and measured depths.

2.4.3 3D Depth Warping

Depth information obtained from the depth cameras are projected to real world coordinates by depth warping. This is achieved by multiplying the depth information with intrinsic characteristics matrix of the depth camera. Each pixel is represented as $p_s(p_{sx}, p_{sy}, D_s)$, where p_{sx}, p_{sy} are (x, y) position and D_s is the disparity at this point. The position in real 3D world $P_s(p_{sx}, p_{sy}, p_{sz})$ is:

$$P_s = K_s^{-1}.p_s \tag{2.2}$$

Where K_s is the matrix of intrinsic characteristics of depth camera. A frame of video and its depth image is shown in Fig. 2.6.

Depth Data Enhancement

The depth data from this TOF depth sensor cannot directly be used due to some inherent problems. In order to use the depth data properly, spatial and temporal

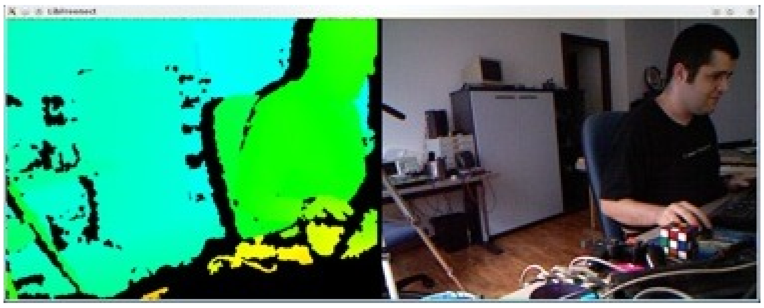


Figure 2.6. Stereo algorithm for 3D mesh generation

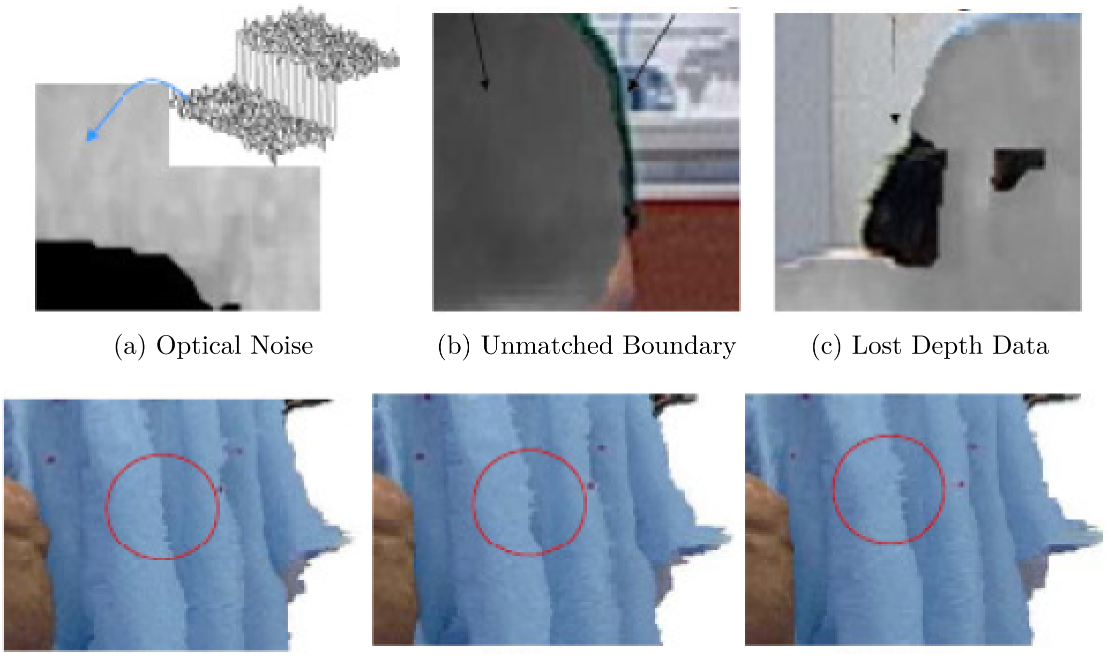
problems existing in the raw depth images needed to be resolved [16], such as:

- Optical noise
- Unmatched boundaries between a depth image and its corresponding color image
- Lost depth data on shiny and dark surface
- Temporal depth flickering artifacts on stationary objects

Optical noise, as shown in Fig. 2.7(a), usually occurs inside of objects in a scene as a result of differences in reflectivity of an infrared sensor according to color variation. Moreover, as shown in Fig. 2.7(b), the depth information is also not registered well with its corresponding color information such as the region of shoulder of a person in Fig. 2.6. The problem of unmatched boundaries arises because the TOF depth sensor exhibits inaccurate behavior at very close and very far target distances. In addition, as shown in Fig. 2.7(c), the TOF depth sensor does not capture depth data well on shiny and dark surfaces such as a black hair region, because reflected lights from these surfaces are very weak or scattered. Especially, as shown in Fig. 2.7(d), these spatial problems cause to generate depth flickering artifacts on a stationary object in the temporal domain. To combat these problems several techniques have been implemented in [16].

2.4.4 Outer Bounary Matching

For the outer boundary selection, they create a trimap of binary depth image and exact outer boundaries which are measured by applying alpha matting on color



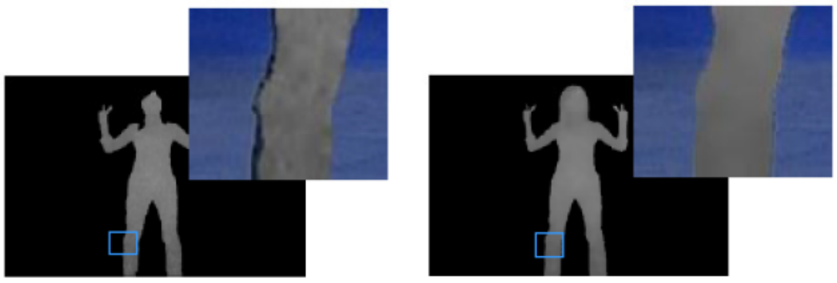
(a) Optical Noise (b) Unmatched Boundary (c) Lost Depth Data

(d) Temporal Flickering

Figure 2.7. Inherent problems of TOF sensors

images. Alpha map is used to compensate depth information with extracted values of depth image by this equation:

$$D_i(x, y) = \frac{A_i(x, y)}{255} * D_i(x - n, y - m) \tag{2.3}$$



(a) Original (b) Boundary matching

Figure 2.8. Outer boundary matching

where $D_i(x, y)$ is the intensity of pixel position (x, y) on the i^{th} depth image, and

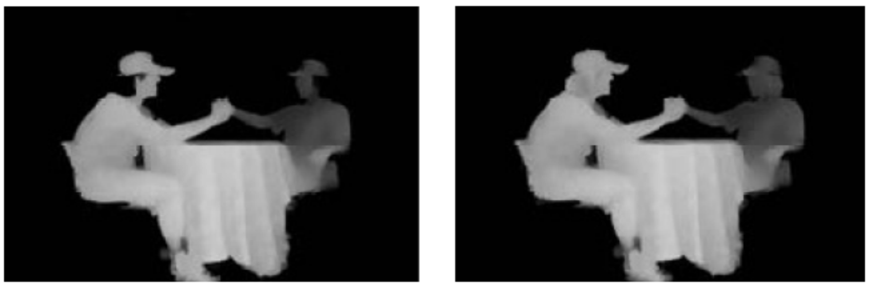
$A_i(x, y)$ is the alpha value on the ith alpha map. The term of $D_i(x - n, y - m)$ is the nearest intensity of $D_i(x, y)$ that is found by a spiral search method. Result of boundary matching is shown in Fig. 2.8.

2.4.5 Optical Noise Minimization

A general bilateral filter reduces noise in an image while preserving important sharp edges. For reducing optical noise inside of objects in depth images, joint bilateral filtering in which edges from both color and depth images are used. In joint bilateral filtering, the assumption is that regions of depth discontinuity in a depth image usually correspond to the edges in its color image.

2.4.6 Recovery of Lost Depth Data

The depth recovery algorithm consists of three steps: detection of the lost depth data region, recovery of the boundary, and estimation of the lost depth data. A region growing algorithm with multiple seeds is applied to detect the lost depth data regions. Then, it recovers the boundary from the detected regions using boundary tracing. Finally, the lost depth data region is filled with depth information interpolated by a quadratic Bézier curve with neighboring depth data on the depth image. Fig. 2.9(b) shows a depth image recovered by the quadratic Bézier curve method from the depth image in Fig. 2.9(a).



(a) Original (b) Depth recovered

Figure 2.9. Depth recovery

2.4.7 Temporal Consistency

Temporal consistency reduces temporal depth flickering artifacts on stationary objects in a scene. For temporal consistency, stationary regions are detected. Block matching is used to estimate the stationary regions of the *t*th frame color image C_t to the *t-1*th frame color image C_{t-1}. Block matching predicts the movement of objects in a scene by estimating similarity between blocks in the temporal domain. After measuring motion vectors $M_t(x, y)$, which is one if block is stationary and zero if block is moving, stationary region is calculated as:

$$S_t(x, y) = D_t(x, y) \& M_t(x, y) \tag{2.4}$$

2.5 Proposed Reconstruction System

The reconstruction system that we want to develop, is an easy to setup system destined for a remote collaboration of virtual avatars. The avatars bodies are prebuilt, while the faces of avatars need to be photorealistic as of participating human players. The features of our proposed system are outlined:

- Minimum setup apparatus should be required, may be a webcam with embedded depth camera, which can be easily installed at personal computer.
- A good refresh rate should be achieved, while gestures and postures are rendered in real-time so that sense of presence is achieved.
- Not necessarily entire user appearance need to be captured, rather only front side of conferee can be captured while avatars can be used to approximate the full body.
- Processing and bandwidth requirements should be limited to common available processor/network capacity.

To achieve these goals, we used Microsoft Kinect for acquisition of data as this device is a very commonly available. We reconstruct only the face of a player, so that it can be attached to a virtual avatar participating in the immersive conference. We transform this face object to common 3D format, compress it and send to a remote client. Reconstructed face object need to be transmitted once in a session, while in rest of time only animation data is needed. In this section we only focus on reconstrunction aspect of the overall system, while the other components are discuessed in subsequent chapters.

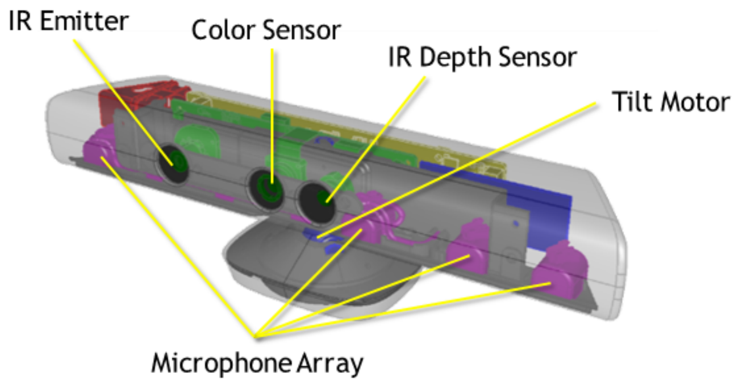


Figure 2.10. Microsoft Kinect Structure

2.5.1 Micrsoft Kinect

The Microsoft Kinect module provides an interface to the Microsoft Kinect XBOX 360 sensor. Based around a webcam-style add-on peripheral, it enables users to control and interact with the Xbox 360 without the need to touch a game controller, through a natural user interface using gestures and spoken commands. The Kinect sensor is a horizontal bar connected to a small base with a motorized pivot as shown in Fig. 2.10 and is designed to be positioned lengthwise above or below the video display. The device features an RGB camera, depth sensor and multi-array microphone running proprietary software, which provide full-body 3D motion capture, facial recognition and voice recognition capabilities.

2.5.2 Color Video Data

Color image data can be accessed in RGB or YUV formats and at different resolution. RGB is presented with 32bit XRGB formatted color bitmaps, while YUV is presented with 16-bit gamma corrected linear UYUV-formatted bitmaps. At regular quality, the Bayer color image data that the sensor returns at 1280x960 is compressed and converted to RGB before transmission to the runtime. The runtime then decompresses the data before it passes the data to application. The use of compression makes it possible to return color data at frame rates as high as 30 FPS, but the algorithm that is used leads to some loss of image fidelity. At high quality, color image data is not compressed in the sensor; it is transmitted to the runtime as originally captured by the sensor. Because the data is not compressed, more data must be transmitted per frame and the maximum frame rate is no more than 15 FPS.

2.5.3 Depth Data

The depth data stream provides frames in which each pixel indicates the distance, in millimeters, to the nearest object at that particular x and y coordinate in the depth sensor's field of view. Depth data streams are available in 640*480, 320*240, 80*60 pixels frame size. The object range should be from 4 to 11 feet away from sensor. If the objects are too close to sensor, depth information will be either empty or filled with noisy values.

2.5.4 Data Acquisition and Pre-processing

We are using Point Cloud Library (PCL) for getting data from Kinect and processing it. PCL is a standalone, large scale, open project for 2D/3D image and point cloud processing. A pointcloud is a frame of 3D points, where each point is represented in 3D real-world projected coordinated. The origin of these 3D points is the camera sensor.

The raw data taken from Kinect is noisy, and many points are missing from the scene. We filter the data to remove noisy scattered points, while we use multiple frame for temporal smoothing of raw data. In particular five consecutive frames are captured to make a pointcloud more suitable to be processed for a better reconstruction results. We detect the face position of the player, and right now we assume there is only one player in front of the camera. Since we are interested only with the reconstruction of face of participating player, so we discard rest of scene, except the face area in both image and depth streams. This small piece of pointcloud is our input to reconstruction algorithm, which generates a mesh from it. The generated mesh have only geometry and connectivity information, so it is mapped to RGB data that we acquired from the face area.

2.5.5 Greedy Projection Triangulation

The Greedy Projection algorithm works by maintaining a list of points, called 'fringe' points, from which the mesh can be grown based on incremental surface growing principle. The algorithm starts from a starting triangle and keeps on adding new triangles until all the possible points are connected [20]. Triangulation is performed locally, by projecting the local neighborhood of a point along the point's normal and connecting unconnected points. Given an unorganized 3D pointcloud, greedy projection algorithm reconstructs the underlying surface's geometrical properties using data resampling and a robust triangulation algorithm in near realtime. A breif flow of greedy projection algorithm is given as:

- Nearest neighbor search: For each point p in the pointcloud, select k neighbors.

  - Nearest neighbors are searched using kd-tree (k dimensional tree). With a reference point p, k neighbors are searched which lie in a sphere of radius r.

- The points in pointcloud are labeled with different states based on their interaction with algorithm:

  - Initially all the points are in *free* state.

  - A point is labeled as *completed*, if all the incident triangles are found.

  - The points that have not yet been chosen as a reference point, are called *fringe* points

- Find normal vectors of each point

- Neighbors are projected on a plane that is approximately tangential to the surface formed by the neighborhood and ordered around p.

- The points are pruned by distance criterion:

  - Candidate adjacent points are searched in the spatial proximity of current reference point using kd-tree. The points which lie outside the sphere of influence centered at reference point are rejected, while the other points are referred to as the candidate points.

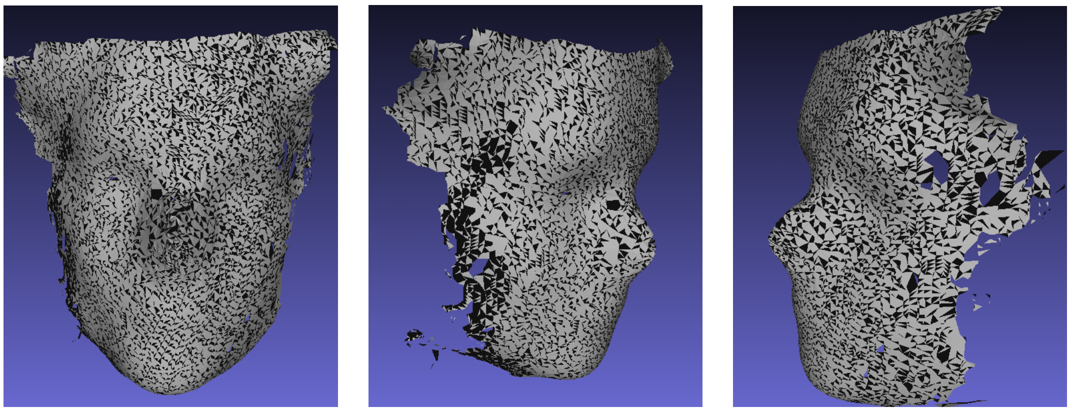

(a) (b) (c)

Figure 2.11. Different views of reconstructedmesh

- Projection plane: the candidate set of points obtained after applying the distance criterion are projected on the the approximate tangent plane.
- Angle ordering: A local coordinate system is defined with the reference point as the origin and the plane projection of the previous step serves as the xy-plane. All the points in the candidate set are projected this plane. Ordering around the reference point is based on the angle θ between the x-axis of the local coordinate system and the vector from origin to the projected candidate point.
- Visibility: the points which potentially can form a self-intersecting mesh are discarded. The algorithm defines two edge types for checking this condition: i. Boundary Edge: an edge with only one triangle incident on it. These edges connect fringe and/or boundary points. ii. Internal Edge: connect the completed points with any other points. The plane is projected using the reference point, candidate set of points and the boundary edges. In case, the line of sight from the reference point to a candidate vertex is obstructed by an edge, the point is occluded.

Using the greedy projection algorithm we reconstruct a 3D object of the face. This model is entirely reconstructed from the data given by Kinect, and no further priory information is used while processing, such as face models used in other state-of-art solutions. Reconstructed model's geometry is almost identical to the player as

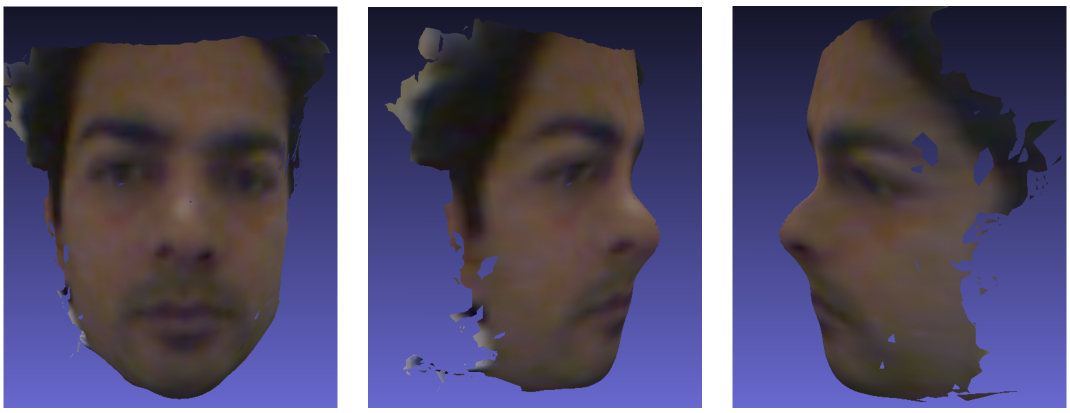

(a) (b) (c)

Figure 2.12. Different views of reconstructedmesh

can be seen in Fig. 2.11. In some experiments we found few holes due to occulusion from the sensor. The perfect solution to fill these holes is to use array of Kinect so that they can capture the scene from different views. But since our required setup needs to be very simple, we have avoided the use of such array of Kinect devices, rather we have interpolated such missing regions to fill up as much holes as possible.

2.5.6 Texture Mapping

The result of greedy projection is a mesh with 3D geometry and connectivity, there is still no color information associated with it. In section 2.5.4 we get RGB data related to face area. We project the triangles produced by greedy projection, and associated the RGB coordinates with each vertex position of the triangle. The resulting texture-mapped mesh can be seen in Fig. 2.12

Chapter 3

Graphics Compression

Interactive 3D graphics plays an important role in many different fields like manufacturing, architecture, entertainment, training, engineering analysis and simulation. It promises to revolutionize many aspects of human-computer interaction. For many of these applications, 3D data sets are increasingly accessed through Internet. The size and complexity of these 3D models is growing rapidly, due to improved design and model acquisition tools, to the widespread acceptance of this technology, and to the need for higher accuracy.

Although many representations [26] have been proposed for exchanging and viewing 3D data sets, but polygonal mesh gained much attention. While the popular graphics libraries e.g. OpenGL, VRML supported the mesh representation for rendering of 3D contents. A mesh can be defined as set of vertices, edges, faces together with their incidence relationships. Meshes are used to represent static or dynamic 3D objects, where the surface of object is approximated by a set of polygons. Approximation error can be high unless number of polygons are sufficiently large as shown in Fig. 3.1. While, on the other hand, large number of polygons are expensive in terms of storage and transmission. Techniques for mesh compression are important in order to make storage and transmission more feasible, in particular when the data is to be streamed in real-time.

Static 3D meshes consist of two types of data: connectivity, which describes the incidence relationship of mesh vertices-faces-edges, and geometry, which assigns 3D locations to vertices. Dynamic 3D meshes are in their generic form represented as a sequence of static meshes with a common connectivity called frames. They require even several times more storage than a single static mesh, unless frames are compressed. Static as well as dynamic meshes show topological as well as geometrical dependencies in spatial and spatio-temporal domain, respectively, which are exploited for compression. Most mesh compression techniques handle connectivity and geometry compression separately.

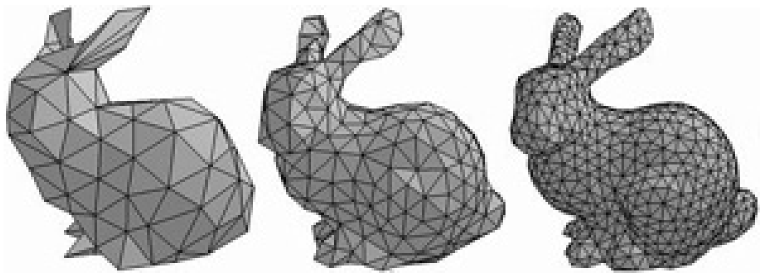


Figure 3.1. A polygonal mesh representation of 3D object

3.1 Geometry Compression

Mesh geometry refers to the set of vertex positions of the graph. The earliest and still most popular method involves a two-stage process of quantization and predictive encoding. Quantization reduces the range of the data. In the context of geometry compression, quantization takes the three components, (x, y, z), of each vertex and stores them in a fixed number of bits (typically 10-14 is sufficient). The quantized mesh at 10-14 bits is visually indistinguishable from the original.

The Parallelogram Predictor was first presented by Touma and Gotsman [31]. It is the most commonly applied predictor in the literature for Single Resolution Compression of vertex locations. It is based on the observation that adjacent triangles tend to form parallelograms, therefore it predicts the next vertex to form a parallelogram with the previous three vertices. Fig. 3.2 illustrates this scheme. To encode the vertex V_4, the authors consider the triangle V_1, V_2, V_3 already coded, and suppose that the polygon V_1, V_2, V_3, V_4 defines a parallelogram, thus they build the vertex V_4'. Then the vertex V_4 is encoded only by its difference vector d with V_4'. [11] generalized this technique, initially adapted for triangular meshes, for arbitrary polygonal meshes. The drawback of these methods is that they cannot be

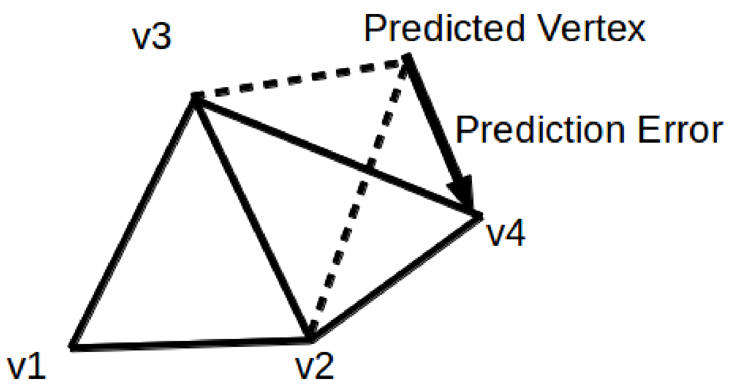


Figure 3.2. Parallelogram Predictor

optimal since the order of enumeration of the vertices is dictated by the coding of the connectivity.

3.2 Connectivity Compression

Connectivity encoding techniques attempt to reduce the redundancy inherent in many popular representations of polyhedral or triangular meshes in 3D. Consider a triangular mesh of V vertices and T triangles, where each triangle is represented by 3 vertex references. The connectivity of each triangle will require storing 3 addresses per triangle (approximately 6 addresses per vertex). So, this scheme will require approximately 60bits per vertex for even a small set of vertices(less than 1024).

3.3 Deering's Geometric Compression

Triangle strips provide a way to represent vertices in fewer bits, where a triangle is formed by combining a new vertex description with the descriptions of the two previously sent vertices, which are temporarily stored in two buffers. Triangle strips only pay off if long strips are build. Deering [8] proposed a triangular strips scheme where a stack buffer of last 16 vertices are used rather than using random access to all the vertices of model. Deering allowed temporarily use of current vertex on stack and reuse of any vertex of last 16 vertices of stack buffer. Deering work reduces the cost to approximately 11 bits per vertex.

3.4 Topological Surgery

Taubin and Rossignac [29] proposed the method Topological Surgery (TS), which was the first method for lossless compression of mesh connectivity and compression of locations of mesh vertices with controllable loss. They decomposed connectivity in vertex and triangle spanning trees, which were encoded. Because proximity in vertex spanning tree often implies geometric proximity of the corresponding vertices, Taubin used ancestors in the tree to predict vertex positions, and thus only need to encode the difference between predicted and actual vertex positions. When vertex coordinates are quantized i.e., truncated to the nearest number in a fixed-point representation scheme, these corrective vectors have on average smaller magnitude than absolute positions and can therefore be encoded with fewer bits.

Linear predictive coding was employed for compression of vertex locations. They were predicted in an order guided by connectivity using already encoded locations. To encode the connectivity, the mesh is first cut through a subset of its edges, called the cut edges. This subset includes all the edges of the vertex spanning tree. The branching nodes and the leaf nodes of the vertex spanning tree are interconnected by vertex runs (i.e., by nodes that have a single child). They compress the representation of the vertex spanning tree by encoding for each vertex run: its length plus two bits of information, which collectively capture the topology of the spanning tree. To increase the compression ratio, they try to build vertex spanning trees with the least number of runs. TS provides the compression to approximately 4bits/vertex. Taubin et al. further extended the TS approach in order to obtain a Levels of Detail representation of a compressed static mesh, providing progressive decoding from low to high resolution. They introduced the Progressive Forest Split (PFS) [28] scheme using a forest, i.e., a set of trees, in order to describe connectivity in different resolution levels.

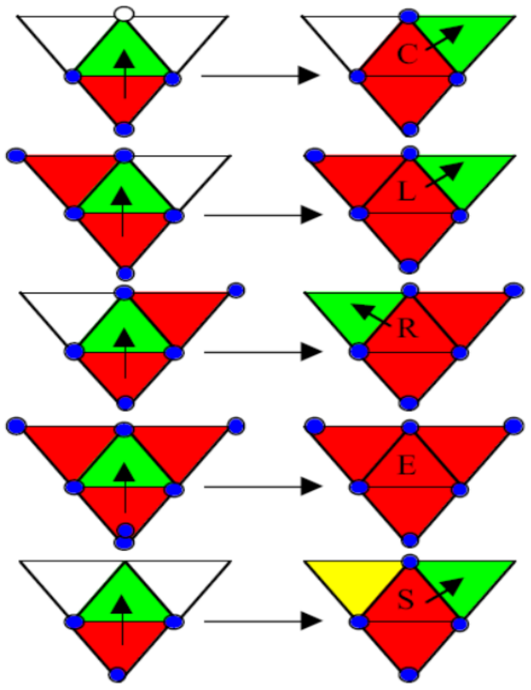


Figure 3.3. EdgeBreaker compression

3.5 EdgeBreaker

EdgeBreaker [25] is a more sophisticated connectivity compression algorithm. EdgeBreaker is a face-based compression scheme which traverses the faces (triangles) of the mesh generating a spanning tree. It uses a finite state machine to compactly describe mesh connectivity. The algorithm encodes one of five symbols at each face to keep the history so that the process can be reversed during decoding. The five symbols form the CLERS string and are defined as follows. A C is encoded when the vertex has not been visited. An L is encoded when the left triangle has been visited, an R is encoded when the right triangle has been visited, an E is encoded when both left and right triangles have been visited and an S is encoded when neither left or right triangles have been visited, as can be seen in Fig. 3.3 In the S case, EdgeBreaker recurses on the right subtree and then the left. EdgeBreaker can compress the connectivity of the mesh to near optimal rates, normally around 3 bits/vertex.

Although EdgeBreaker is a simple, efficient method for compressing the connectivity of a mesh, it has some limitations. First it is limited to triangle meshes. In practice, as many meshes are triangulated, this may not seem a large problem, but nevertheless it limits the areas in which it can be applied. Second, EdgeBreaker requires random access to the vertices. This is inconvenient for gigantic meshes for out-of-core processing.

3.6 FaceFixer

Since EdgeBreaker works only with triangular meshes, FaceFixer [12] is the improvement of EdgeBreaker to work with generalized polygonal mesh. The connectivity of the polygon mesh is encoded as a sequence of labels F_n, R, L, S, E, H_n, and $M_{i,k,l}$. The total number of labels equals the number of mesh edges. The sequence of labels represents an interwoven description of a polygon spanning tree and its complementary vertex spanning tree. For every face of n sides there is a label F_n and for every hole of size n there is a label H_n. Together they label the edges of the polygon spanning tree. For every handle there is a label $M_{i,k,l}$ that has three integer values associated. These specify the two edges of the polygon spanning tree that need to be 'fixed' together to re-create the handle. The remaining labels R, L, S, and E label the edges of the corresponding vertex spanning tree and describe how to 'fix' faces and holes together. Subsequently an entropy coder compresses the label sequence into a bit-stream.

3.7 Geometric Progressive Compression

While in all approaches so far mesh compression is mainly guided by mesh connectivity, [9] proposed a progressive compression algorithm mainly driven by the geometry. They recursively subdivided the bounding box of the object into smaller and smaller cells, until they contain, at most, one vertex. At each subdivision level, they transmit only occurrences of points (the authors demonstrated that it is enough to reconstruct a point set). Concerning connectivity, they encoded vertex splits which occur at each subdivision step. The authors have later on generalized this approach, which gives excellent results, to arbitrary non-manifold meshes.

3.8 MPEG-4 Graphics Compression

The MPEG committee had a continuous activity on compressing 3D graphics assets and scenes since the first version of MPEG-4 was published. MPEG-4 offers many tools that can be classified with respect to data type. Binary Format for Scene Description (BIFS) is one of these tools that allows compressing a scene graph. A scene graph is composed of many different type of nodes, so a tool to compress a scene graph is made generic. Due to its generic nature, BIFS can not completely exploit the redundancy of specific data such as meshes or animation curves. Lately, MPEG-4 defined specific tools for compressing each type of data independetly. The tools for mesh compression comprises 3D Mesh Coding (3DMC) and Wavelet Subdivision Surface (WSS), for animation Coordinate Interpolator, Orientation Interpolator and Position Interpolator (CI, OI, PI), Bone-Based Animation (BBA) and Frame-based Animated Mesh Compression (FAMC) and for appearance MPEG-4 natively supports PNG, JPEG and JPEG2000. Here we will explain only 3DMC and its extension Scalable 3DMC (SC3DMC), which deal with compression of meshes.

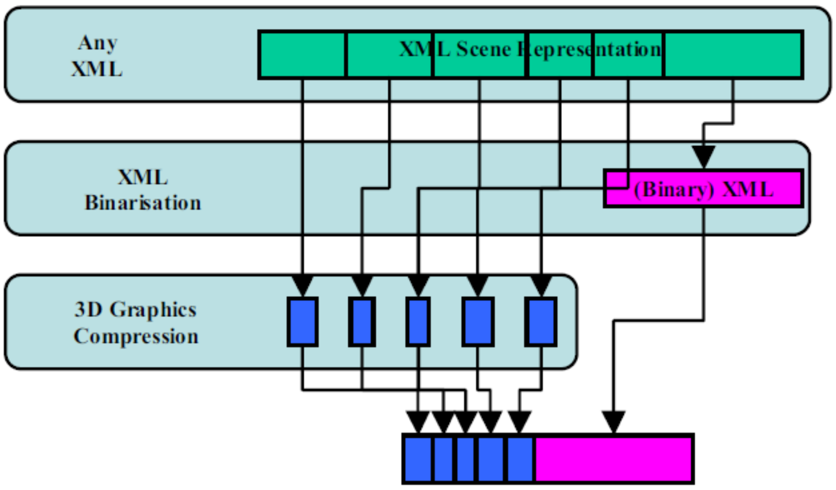


Figure 3.4. Layers of 3D graphics compression model

While XML-based reprsentation of 3D data is appropriate for data exchange in the production phase, the verbosity and redundancy of XML make it highly inappropriate for distribution to end users. Considering this limitation, the MPEG committee initiated an activity with the goal of applying 3D graphics compression on XML-based scene representation. The activity led to the publication of a new part of the MPEG-4 standard, Part 25 [14]. The standarad specifies an architectural model, which works with a third party XML description of the scene graph with (potential) binarization tools and with MPEG-4 3D graphics compression tools. The advantages of this approach is the use of powerful compression tools for graphics and the generality of graphic primitives representation. MP4 formatted bitstreams are obtained using the proposed architectural model. Advanced compression features such as progressive decompression and support for streaming, provided by the specific compression tools, are preserved with the new model, since it decouples the decoding of the scene graph from the decoding of the object graph. Typically, when consuming content compliant with MPEG-4 Part 25, a player builds first the scene graph based on XML description, connects the decoders of the elementary streams and renders the results based on the time stamps of the elementary stream. There is no need to rebuild the original XML file before rendering. As illustrated in Fig. 3.4, MPEG-4 Part 25 defines an architectural model that have three layers: textual data representation, binarization and compression.

3.9 Scalable Complexity 3D Mesh Coding

MEPG introduced 3D Mesh Coding (3DMC), which is based on Topological Surgery (TS) representation [30], it works with the meshes which are defined by indexed list of polygons, and consists of geometry, topology and other properties of vertices. In this scheme the connectivity of the mesh is encoded with no loss of information, and the vertex positions and properties are quantified and hence encoded with variable loss of information. 3DMC Extension (3DMCe) was published in MPEG-4 Part 16 [13], which added support for efficient texture coordinate compression and mesh animation. Later on as an amendment, the Scalable Complexity 3D Mesh Compression (SC3DMC) [15] tool set was added which provided three encoding methods for mesh compression. as briefly discussed in following sections. Fig. 3.5 shows a block diagram of SC3DMC.

3.9.1 Triangle Fan-based Compression

Triangle Fan-based compression (TFAN) [19] deals with 3D meshes of arbitrary topologies, while offering a linear computational complexity (with respect to the

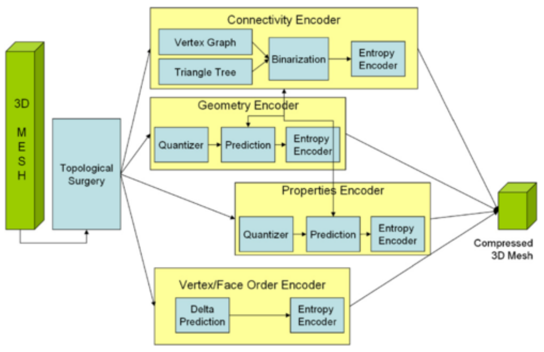


Figure 3.5. Block diagram of SC3DMC

number of mesh vertices) for both encoding and decoding algorithms. In addition, the TFAN compressed representation is optimized for real-time decoding applications.

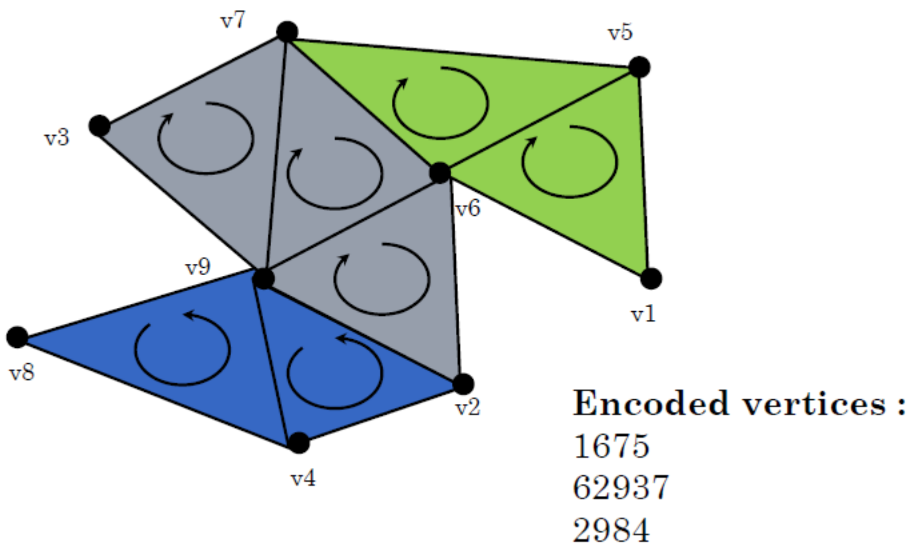


Figure 3.6. Encoding Vertices with TFAN

The TFAN encoder traverses the meshes from neighbor to neighbor. In the beginning all the vertices are labelled as non-visited, then starting from a vertex each successive vertex is traveresed. At each stage a traingle fan is created which is incident to current vertex and have minimum number of neighbor trainalges. Triangle fans are made for all possible vertices intially labelled as non-visited, and at each stage the visited vertex is marked so that not to repeat in the next stage. When all of the vertices are traveresed and tiangle fans are made, the vertices are encoded with the triangle fan's order, hence the vertices which are shared inside trinagle fan are saved to encode as shown in Fig. 3.6.

TFAN outperforms about 20 % when compared to previous techniques in 3DMC. But as compared to [31] it reduces to compression efficiency by 10 %, this is the price to pay the generality of the TFAN representation, which is specifically designed for dealing with generic, both manifold and non-manifold meshes.

The TFAN decoder reconstructs the mesh connectivity by successively decoding the set of triangle fans transmitted. The list of triangle fans are traversed and mesh connectivity is generated by the incident relationship of triangle fans.

3.9.2 Shared Vertex Analysis

Share Vertex Analysis (SVA) [27] encodes the connectivity information by exploiting the shared nature of a vertex. Since a vertex may be used by several faces, therefore, it is very likely that the vertices used in the previous face can also be used in the current face. Therefore, using SVA encoding mode, the connectivity between the current face and the previously encoded face is checked by counting the number of shared vertices. SVA presents a fast 3D mesh compression method that compresses the redundant shared vertex information using simple first-order differential coding followed by fast entropy coding with a fixed length prefix.

The connectivity between the current face and the previously encoded face is checked by counting the number of shared vertices. The number of shared vertices varies from zero to three, if an input 3D mesh is triangular. Therefore, SVA defines four different modes to represent vertices for the current face as shown in Fig. 3.7. For a triangular face the three modes depend on the shared vertices with previous face:

- **Mode 0** the vertices in the previous face are not shared in the current face.
- **Mode 1** one vertex in the previous face is shared in the current face.
- **Mode 2** two vertices in the previous face are shared in the current face.
- **Mode 3** all vertices in the previous face are shared in the current face.

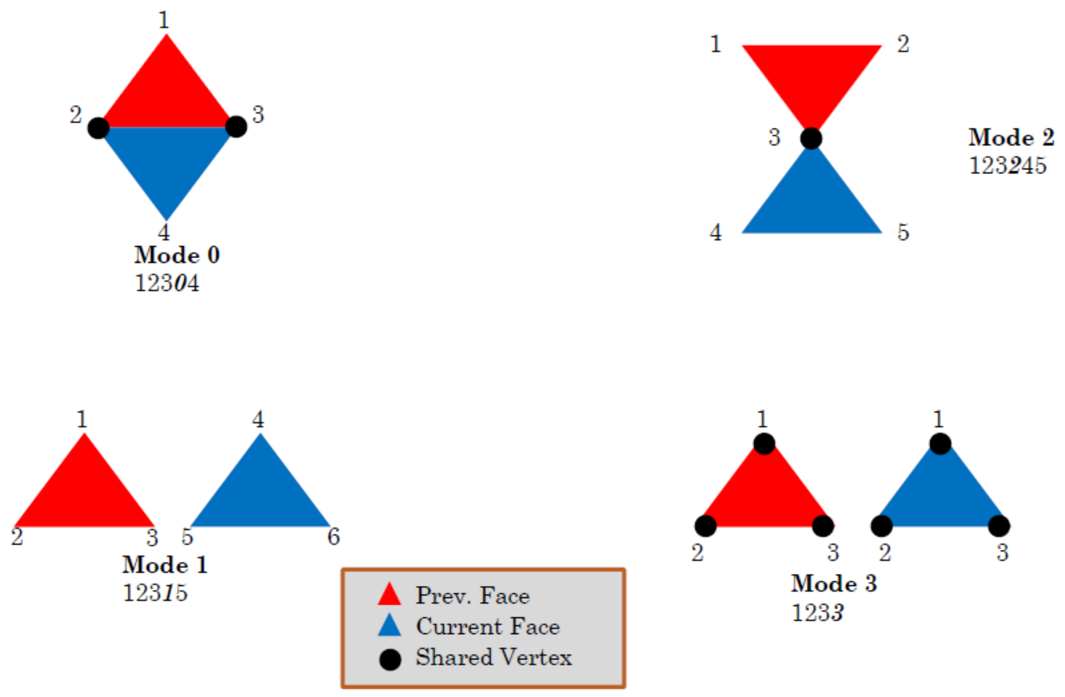


Figure 3.7. Encoding Vertices with SVA

Indices of new vertices in modes 0, 1, and 2 are subject to differential encoding by computing the difference in the vertex index, which is calculated by comparing the current vertex index and the previous vertex index.

3.9.3 Quantization Based Compact Representation

Quantization-Based Compact Representation (QBCR) [18] is computational efficient encoding method that encodes the geometry and photometry by applying only quantization without any prediction. After quantization, binarization is applied directly on the quantized values without any entropy coding. The connectivity information are encoded without quantization because lossless compression is required to preserve the toplogy of model.

3.10 Graphics Compression Test

We tested graphics sequence of large size, that really needed to be compressed in order to efficiently transmit them over network. We used a machine with Intel(R) Pentium(R) 4 CPU 3.00GHz, and a RAM of 4GB. The test sequences are full human body scanned 3D meshes, that have some million of vertices (2 millions), and each test sequence have the size more than 30MB. In the results we will show compression efficiency and complexity of only two testcases: Testcase1 of 40 MB, and Testcase2 of 36MB.

3.10.1 MPEG Graphics Encoder

We worked with MyMultimediaWorld (MMW) graphics encoder, which is an optimized open-source implementation of MPEG-4 Part 25 [14]. It implements the encoder and decoder of popular XML-based scene graph formats (COLLADA, X3D and XMT), and the elementary supported techniques for scene graphs compression is SC3DMC, while for animation stream it provides Bone Based Animation (BBA) compression. The encoder takes XML like format as input, for each format it has a parser that parses all of the components (geometry, connectivity, texture etc...). Then each component is compressed using graphics encoder, and finally all compressed streams are combined together to generate a single MP4 bitstream. SC3DMC supports popular techniques for mesh compression i.e. TFAN, SVA and QBCR as been described in Section 3.9. Different options can be choosen as per requirement are given below:

- **EncodingMode:** QBCR, SVA, TFAN
- **BinarizationMode:** FL, BPC, 4C, AC, AC/EGC
- **QuantizationParameter:** 0 31
- **PredictionMode:** No Prediction, Differential, XOR, Adaptive, Circular, Fast Parallelogram

3.10.2 Results

We tested the performance of encoder in terms of encoding gain, encoding complexity and surface distortion. We varied the bits/componect (b/c) values, which is the number of bits to be assigned to represent each component. We selected the range of b/c as 6-15, because lower than 6, the surface is too distorted to mimic with the original. On the other hand, when we go higher than this range we don't achieve any better surface but we loose the compression efficiency.

3.10.3 Encoding Gain

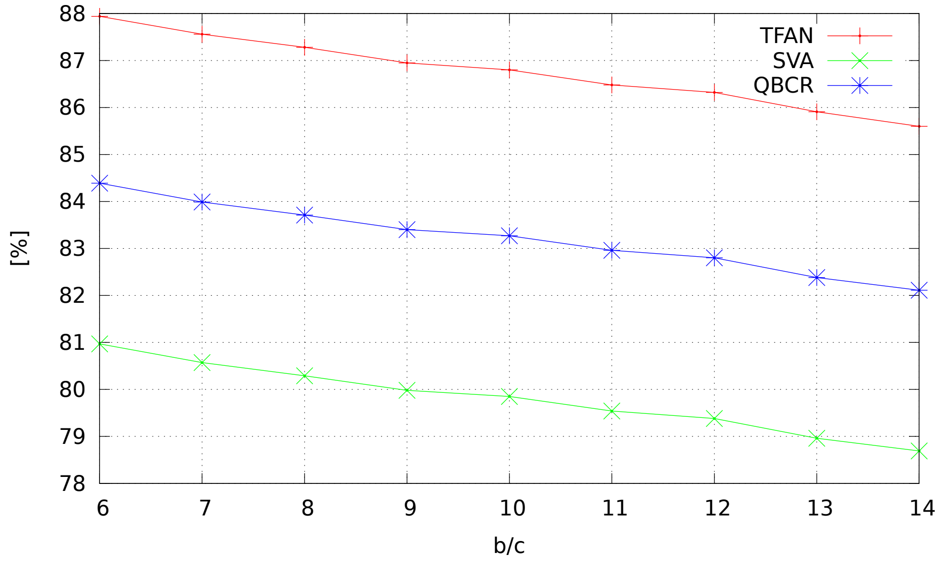
Encoding gain of two test cases are given in Fig. 3.8. In all the compression methods, efficiency is always more than 80%, but we can observe that TFAN gives us the highest compression gain, even with the maximum value of b/c it gives 85% of gain. SVA gives the worst efficiency, while QBCR is always between SVA and TFAN.

3.10.4 Encoding Complexity

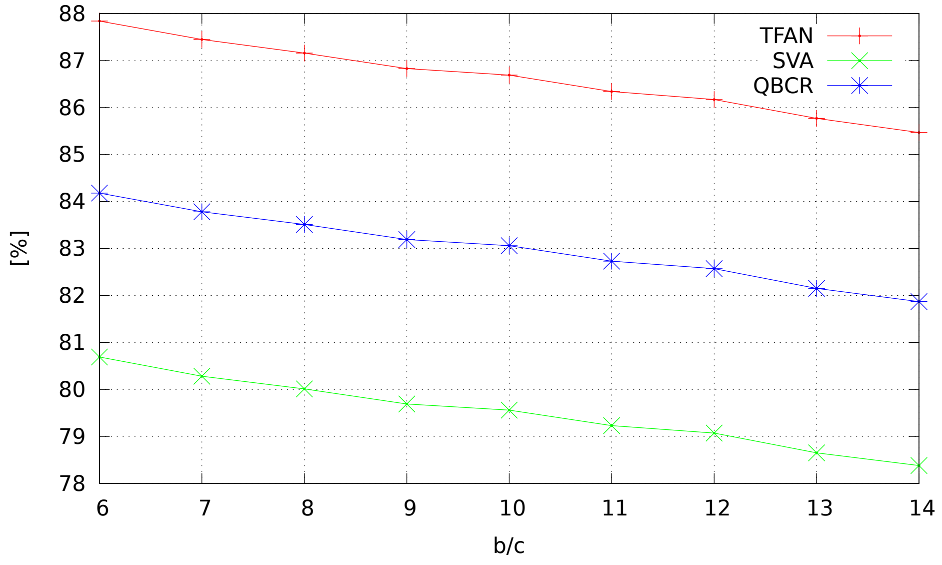
Encoding complexity of our two testcases are given in Fig. 3.9, we can observe that QBCR is lowest in complexity, this is because its encoding process is very simple. On the other hand TFAN is the most complex algorithm to compress. So this is a trade-off between efficiency and complexity, and exact choice to choose between depends on the application.

3.10.5 Surface Distortion

We decoded the encoded objects, and measured the surface distortion between the original and the decoded one. For this measurement we used the MESH tool [5], that computes the Hausdorff distance between two meshes. Hausdorff distance is the maximum distance of a set to the nearest point in the other set, so for polygonal it computes the maximum distance of polygons which describes how much a surface is differed to the other surface. The result of this measurements are given in Fig. 3.10, where we provided the RMs error between the two surfaces for the case of TFAN.



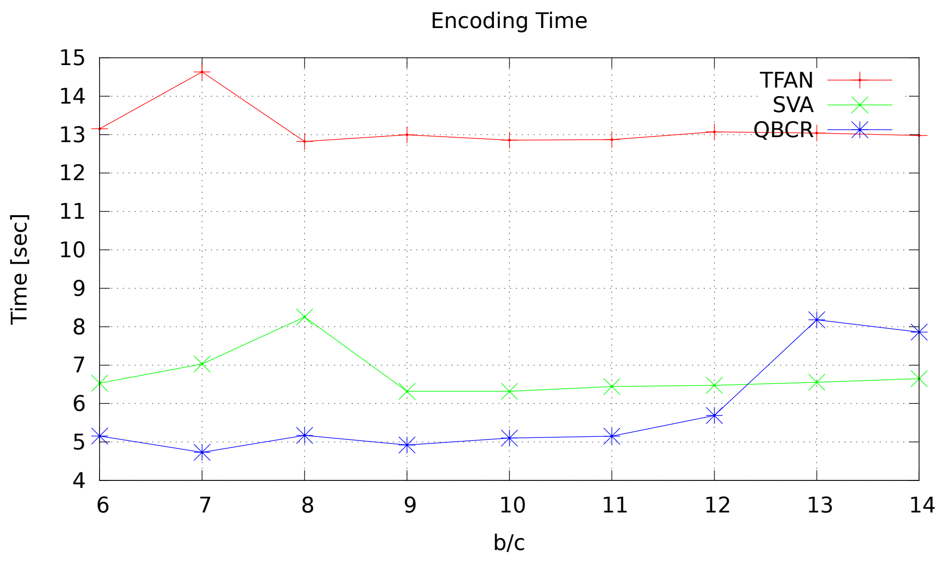
(a) Testcase1



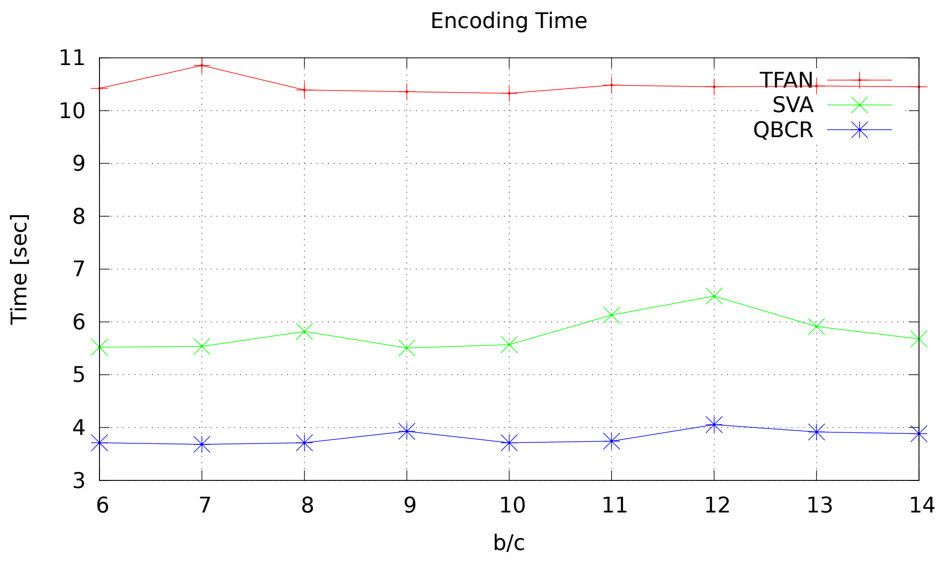
(b) Testcase2

Figure 3.8. Encoding gains

It can be observed that decoded surface faces some distortion when b/c is less than 10, and at each point it goes towards zero, while for all values higher than 10, the distortion RMS is about zero. Hence b/c higher than 10 always gives the identical decoded surface. The same can be observed in inspecting the decoded



(a) Testcase1



(b) Testcase2

Figure 3.9. Encoding complexity

surfaces visually in Fig. 3.11 and Fig. 3.12.

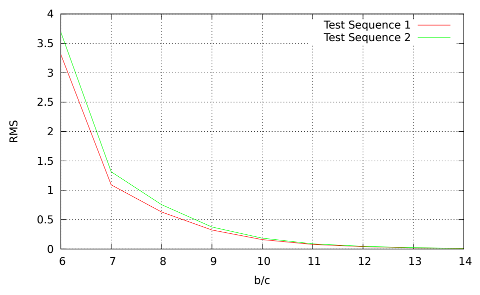
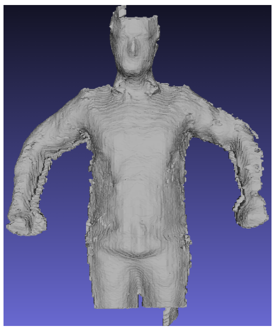
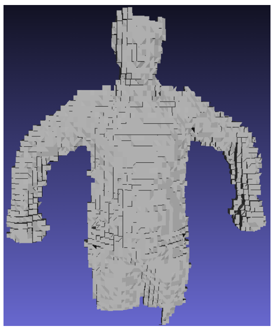
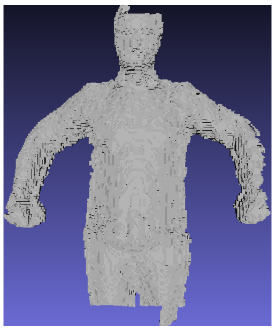
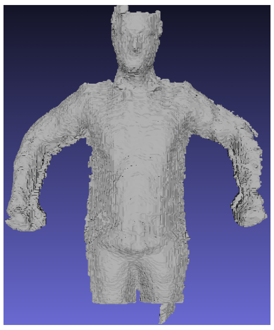


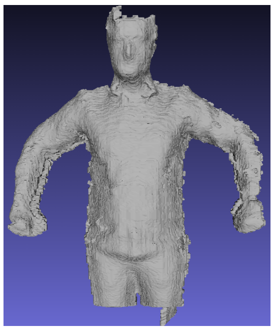
Figure 3.10. Hausdorff distance between decoded and original surfaces

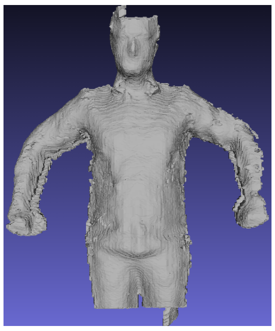

(a) Original

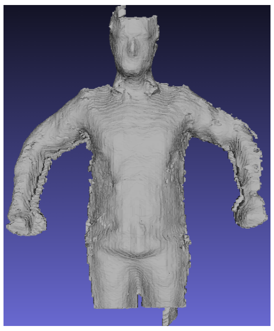

(b) b/c = 6

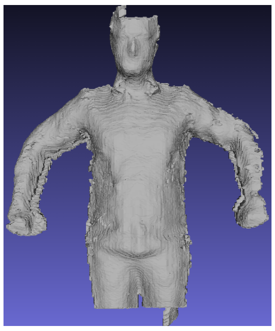

(c) b/c = 7

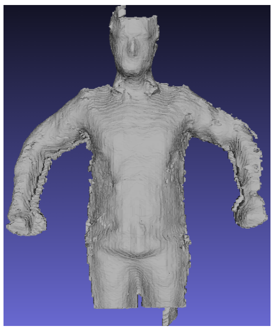

(d) b/c = 8


(e) b/c = 9


(f) b/c = 10


(g) b/c = 11


(h) b/c = 12


(i) b/c = 13

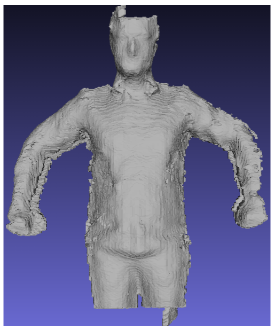
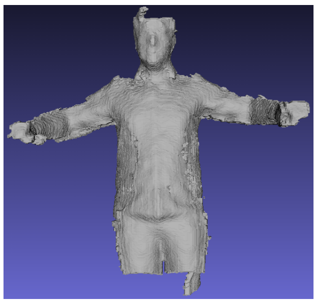
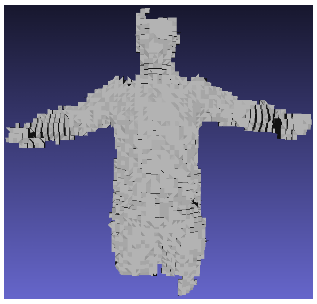
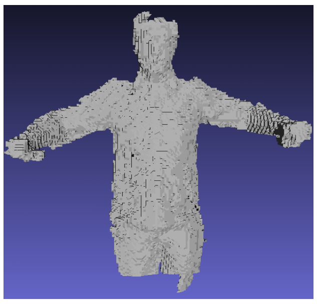
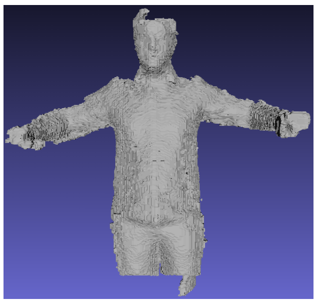

(j) b/c = 14

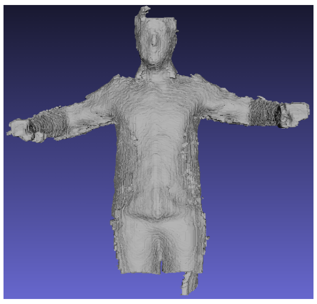
Figure 3.11. Visual inspection of decoded surfaces of Testcase1

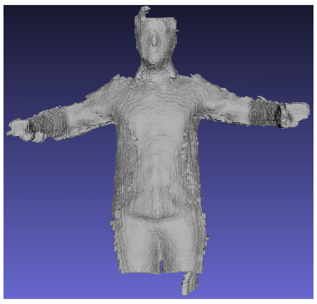

(a) Original

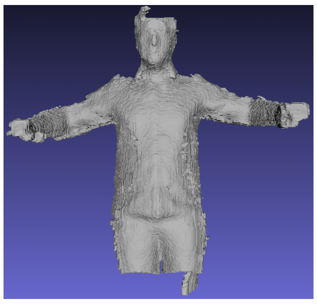

(b) b/c = 6

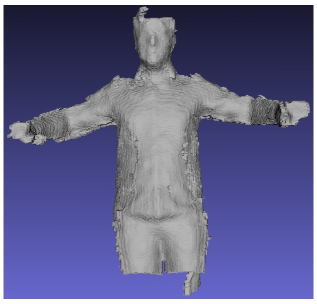

(c) b/c = 7

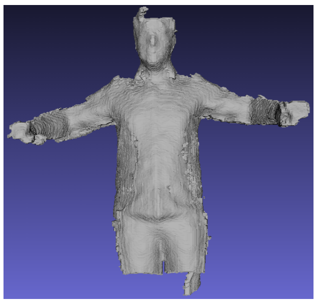

(d) b/c = 8


(e) b/c = 9


(f) b/c = 10


(g) b/c = 11


(h) b/c = 12


(i) b/c = 13

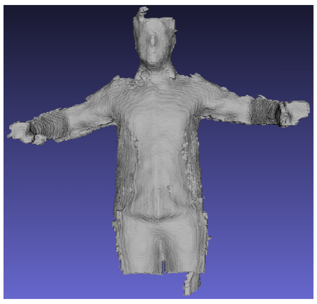

(j) b/c = 14

Figure 3.12. Visual inspection of decoded surfaces of Testcase2

Chapter 4

Animation and Skeleton Extraction

Human like avatars are being used in many virtual reality applications, and it is very much desired to control the avatar in natural interaction. Such natural interaction movements must be captured from the participating player in the application, and transmitted to avatars which is going to be animated. In the past, MPEG-4 addressed the animation of human avatars by introducing Face and Body Animation (FBA), which described the avatar modelling representation and the animation parameter representation and its compression. The latest activities of MPEG-4 lead to adoption of Bone Based Animation (BBA) that defined a framework to realistically animate a generic articulated avatar. In this chapter, we will first discuss details about BBA framework, then we will present our proposed method to obtain motion data for BBA.

4.1 Bone Based Animation

The goal of MPEG-4 BBA is to define an interchange format where a generic Virtual Character (VC) animation and modeling can be specified. It does not specify any animation method, rather it only defines a low level representation format for articulated generic virtual character. Moving a VC requires to define the deformation behavior, and the minimal set of animation parameters for moving it. BBA natively support the VC geometry in form of polygonal meshes, but in general it can support

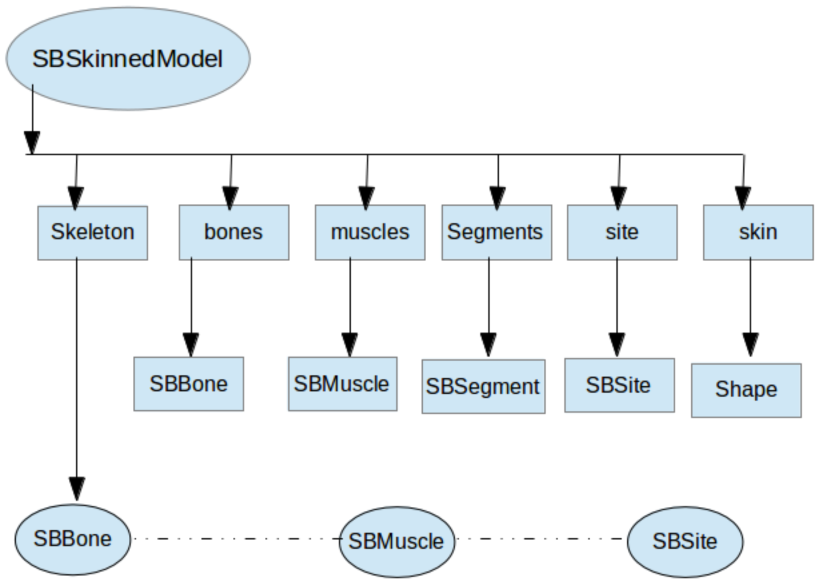


Figure 4.1. SBSkinned node

any kind of geometrical representation, provided that it is able to associate skeleton with geometry.

VC geometry is deformed while applying an animation to it. In literature, there exists many approaches that deal with such deformations, but in scope of BBA here only Skeleton based deformation is discussed. In the framework of BBA, VC is modeled in three layers [24]: Skeleton, Muscle and Skin. The animation data is associated with skeleton, while for each joint on the skeleton there are defined weighted transformations over the skin, so that skin is transformed in coherence to the skeleton. The muscle layer relies on a NURBS-based representation and induces local skin deformations. The animation parameters of skeleton are expressed in individual component of rotation, normally formed by joints. The deformations on these joints are elementary rotation, which can be roto-translation matrices or quaternions.

The Binary Format for Scene (BIFS) defines a structure of nodes where heterogenious type of objects can co-exist, and maintain their temporal and spatial relationships. Each object is represented as a standardized node or a set of nodes. A dedicated node for defining BBA in BIFS is designed, which is called SBSkinned-Model, a block diagram of its heirarchy is shown in Fig. 4.1.

The skeleton is a hierarchical structure made of bones. A bone is represented in the scene graph as SBBone node, and it has a unique parent in the hierarchy and can have one or more children. The Children of a bone can be another bone (SBBone), muscle (SBMuscle), specific relative 3D location (SBSite) and 3D object grouping nodes (SBSegment).

Three types of information are associated with SBBone: the effect of bone deformation on skin, the relative generic geometric transform of the bone with respect to its parent, and data related to inverse kinematics. Each bone has an influence on the skin surface. By changing the bone position or orientation, some vertices of the model skin will be affected by a translation component. The SBMuscle node adds local deformation for simulating muscle action on skin.

Skeleton motion is defined by means of individual bone. The bone motion is always specified by a geometric transform with respect to the initial registration of the VC's static position, it can be represented in any form rotation matrix or a quaternion. A bone can be rotated with respect to any axis, can be translated in any direction and can also be scaled to attain required deformation.

4.2 Proposed Skeleton Extraction Method

There are many methods for sekeleton extraction in state of the art, some of them are supervised where many priory known information are required to build the skeleton, e.g. [10] . Others are unsupervised, but they process pointcloud iteratively until skeleton is formed, e.g. [7], hence become very complex.

Our proposed method is very simple and computationally efficient. We use both data provided by Kinect i.e. depthmap as well as RGB data. To start with, we process the location of player, by detecting the face of human player from RGB stream. From this position, we got many other information all associated with it, e.g. approximate size of player, approximate regions to search for other body parts, because all of the body parts are always in coherence with generally assumed human body. Starting from face loaction in the depth map (as they are at the same locations), we find the human contour from depth map as shown in Fig. 4.2. The midlines of this contour is a rough 2D skeleton of player. At this point we segment different body parts as shown in Fig. 4.3, and hence we locate the joints. Then we calculate average depth around joint position and assign the third dimension to already obtained 2D skeleton.

From each frame, we calculate the delta of joint movements, and we write a motion data file that contains the motion information of each joint for every frame. The motion data file is explained in section 4.3.

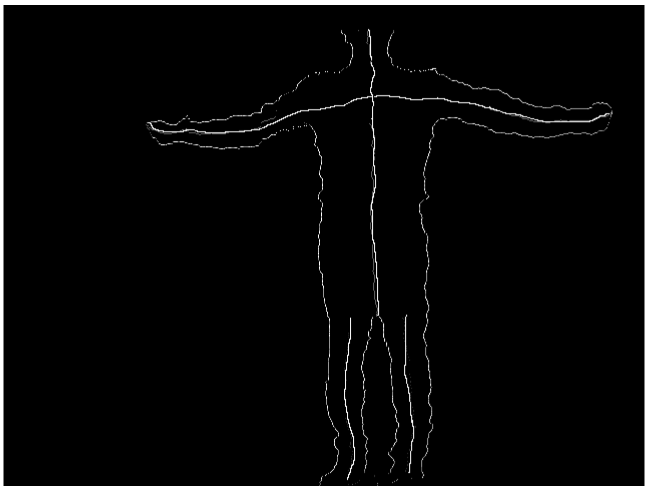


Figure 4.2. Contour Midlines

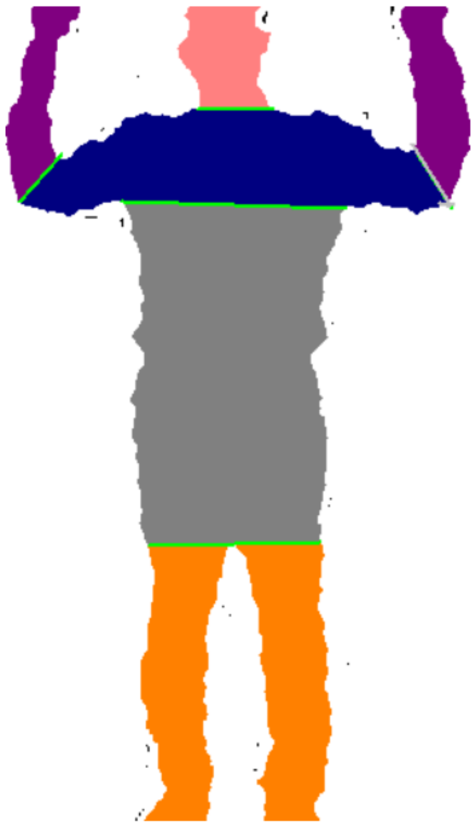
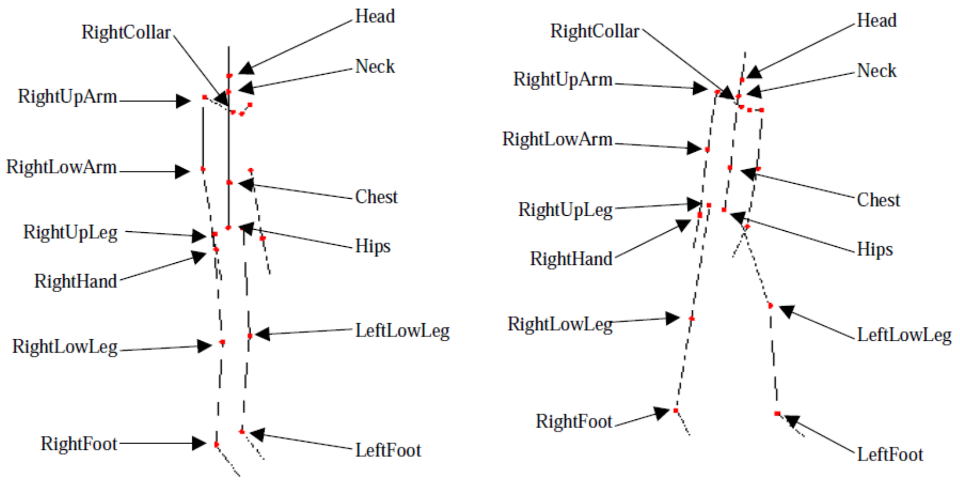


Figure 4.3. Segmentation of body parts



(a) Initial pose (b) First frame motion

Figure 4.4. BVH skeleton pose

4.3 Motion Data

We used BioVision Heirarchical (BVH) format for storing motion data, which is a hierarchical data structure, that can represent the bones of the skeleton. The BVH file consists of two sections: Header and Data. Header section describes heirarchy and initial pose of skeleton, while data section describes the channel data for each frame. Illustrations of the base position and the first frame of an animation are given in Fig. 4.4, while a segment of bvh file is listed in Fig. 4.5.

```
HIERARCHY
ROOT Hips
{
      OFFSET      0.00  0.00  0.00
      CHANNELS 6 Xposition Yposition Zposition Zrotation Xrotation Yrotation
      JOINT Chest
      {
            OFFSET      0.000000    6.275751    0.000000
            CHANNELS 3 Zrotation Xrotation Yrotation
            JOINT Neck
            {
                  OFFSET      0.000000    14.296947   0.000000
                  CHANNELS 3 Zrotation Xrotation Yrotation
                  JOINT Head
                  {
                        OFFSET      0.000000    2.637461    0.000000
                        CHANNELS 3 Zrotation Xrotation Yrotation
                        End Site
                        {
                              OFFSET      0.000000    4.499004    0.000000
                        }
                  }
            }
      }
}
MOTION
Frames: 2
Frame Time: 0.04166667
-9.533684   4.447926    -0.566564   -7.757381   -1.735414   89.207932   9.763572
            6.289016    -1.825344   -6.106647   3.973667    -3.706973   -6.474916
            -14.391472  -3.461282   -16.504230  3.973544    -3.805107   22.204674
            2.533497    -28.283911  -6.862538   6.191492    4.448771    -16.292816
            2.951538    -3.418231   7.634442    11.325822   5.149696    -23.069189
            -18.352753  15.051558   -7.514462   8.397663    2.953842    -7.213992
            2.494318    -1.543435   2.970936    -25.086460  -4.195537   -1.752307
            7.093068    -1.507532   -2.633332   3.858087    0.256802    7.892136
            12.803010   -28.692566  2.151862    -9.164188   8.006427    -5.641034
            -12.596124  4.366460
```

Figure 4.5. A short segment of BVH file

The hierarchical section of the file starts with the keyword HIERARCHY, which is followed on the next line by the keyword ROOT and the name of the bone that is the root of the skeletal hierarchy. The ROOT keyword indicates the start of a new skeletal hierarchical structure and although the BVH file is capable of containing many skeletons, it is usual to have only a single skeleton defined per file. The remaining structure of the skeleton is defined in a recursive nature where each bone's definition, including any children, is encapsulated in curly braces, which is delimited on the previous line with the keyword JOINT (or ROOT in the case of the root bone) followed by the name of the bone.Within the definition of each bone, the first line, delimited by the keyword OFFSET, details the translation of the origin of the bone with respect to its parent's origin (or globally in the case of the root bone) along the x, y and z-axis respectively. The offset serves a further purpose of implicitly defining the length and direction of the parent's bone.

Processing all the header file we can define the skeleton heirarchy. Once the skeletal hierarchy is defined, the second section of a BVH file, which is denoted with the keyword MOTION, contains the number of frames in the animation, frame rate and the channel data. The line containing the number of frames starts with the keyword 'Frames:' which is followed by the number of frames. The frame rate is on a line starting with 'Frame Time:' which is followed by a positive float that represent the duration of a single frame. To convert this into a frames per second format you simply need to divide 1 by the frame time. Once the number of frames and frame time has been defined, the rest of the file contains that channel data for each bone in the order they were seen in the hierarchy definition, where each line of float values represents an animation frame.

The construction of the rotation matrix, R, can be easily done by multiplying together the rotation matrices for each of the different channel axes in the order they appeared in the hierarchy section of the file. For example, consider the following channel description for a bone:

```
CHANNELS 3 Zrotation Xrotation Yrotation
```

This mean that the compound rotation matrix, R, is calculated as:

$$R = R_z * R_x * R_y \tag{4.1}$$

Once the composite rotation matrix is calculated, using a homogeneous coordinate system, the translation components are simply the first 3 cells of the last column (whereas the rotational components take up the top left 3x3 cells), as illustrated in Equation 4.2. Normally, the root is the only bone that has per-frame translation data, however each bone has a base offset that needs to be added to the local matrix stack. Therefore, T_x, T_y and T_z represent the summation of a bone's base position and frame translation data.

$$M = \begin{bmatrix} R_{00} & R_{01} & R_{02} & T_x \\ R_{10} & R_{11} & R_{12} & T_y \\ R_{20} & R_{21} & R_{22} & T_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{4.2}$$

The local transformation of a bone describes its orientation within in its local coordinate system, which in turn is subject to its parent's local orientations. To obtain a global matrix transform for a given bone, the local transformion needs to be premultiplied by its parent's global transform, which itself is derived by multiplying its local transform with its parent's global transform and so on. Equation 4.3 describes this combination sequence, where n is the current bone whose parent bone is $n - 1$ and $n = 0$ is the bone at the root of the hierarchy.

$$M^n_{global} = \prod_{i=0}^{n} M^i_{local} \tag{4.3}$$

Figure 4.6. 3D graphics player showing a troll object

4.4 3D Graphics Player

To visualize the motion captured, we need a system that can run this motion on an avatar. In a fully implemented system, this visualization is required on a remote machine with a player interacting in a game or in virtual meeting. For our purpose, we used a 3D graphics player which is developed inside STMicroelectronics. This player is used to run compressed MPEG-4 objects, so it has builtin decoder for decompressing an mp4 object produced by MPEG-4 encoder. We have a set of avatars that can be given input to the player, a snapshot of player is given in Fig. 4.6.

So, our task is to develope support with graphics player so that it can run the motion data captured by our system. The graphics player works with the absolute animation roto-translation matrices. At each frame it reads the roto-translation data related to each bone, and displays it on its rendering engine, although interpolation are performed for intermediate frames to smooth down the motion of avatar.

In our skeleton extraction process, we get the motion data associated with 16 joints of body, with the heirarchy given as:

- Pelvis
  - LeftHip
    - LeftKnee
      - LeftFoot
  - RightHip
    - RightKnee
      - RightFoot
  - UpperBack
    - LeftUpperArm
      - LeftElbow
      - LeftHand
    - RightUpperArm
      - RightElbow
      - RightHand
    - Neck
      - Head

In the graphics player we are using avatar of a Troll object as can be seen in Fig. 4.6. This troll object has many bones, and its bone formation are given in Fig. 4.7. To properly animate any object, we have to assicate the bones from originating source to destination avatar object if both of the objects are not the same genre, like if the case of human avatar, such assocation can be pre-understood. So, to properly animate the troll object, we found the bones association as given in Table 4.1.

The transformation data generated by us is convereted from heirachical representation to absolute rotation matrices as described in section 4.3. We run graphics player with the generated rotation data, and we received exact animation as being produced by the player.

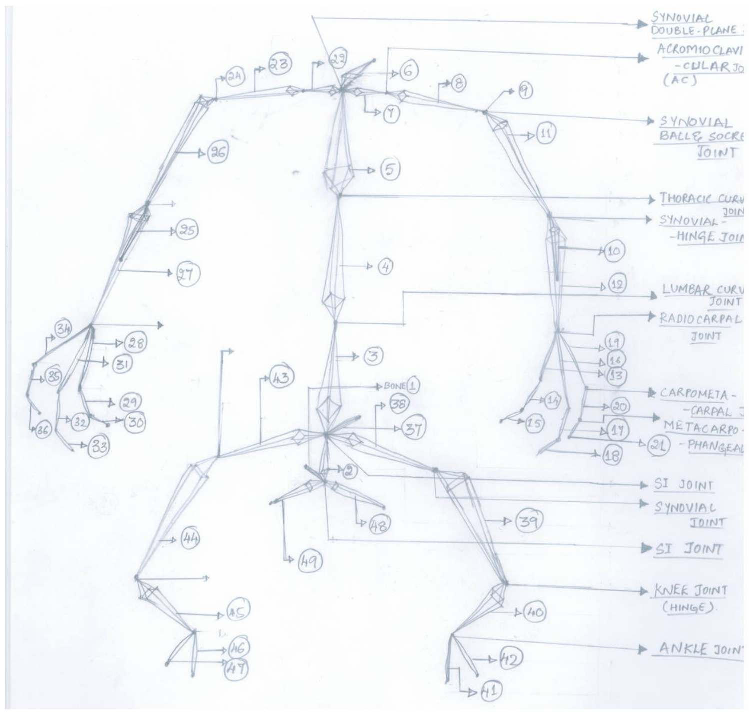


Figure 4.7. Bones of troll object

Table 4.1. Bone mapping between troll and human body

Joint Name	**Bone Number**
Pelvis	3
LeftHip	43
LeftKnee	44
LeftFoot	45
RightHip	38
RightKnee	39
RightFoot	40
UpperBack	4
LeftUpperArm	23
LeftElbow	26, 24
LeftHand	27
RightUpperArm	8
RightElbow	11, 9
RightHand	12
Neck	5
Head	6

Chapter 5

Conclusions

In this work we covered entire pipeline of a 3D immersive system. We implemented an acquisition system that reconstructs a human face. Our target application is a small setup based 3D capturing system that can be deployed easily in a client machine. So, we used Microsoft Kinect to capture 3D data and processed this data to form a mesh of human face.

A performance analysis of MPEG-4 part 25 encoder is presented for user generated human body models, which are huge in size and need to be compressed. We found TFAN the best compression method. With a bits per component value more than 10, compression efficiency is higher than 80% without significant visual loss in quality.

We worked with motion acquisition of a player, we extracted skeleton for 16 joints of human body. The motion data is tested with a 3D graphics player, and we found it to be coherent with actual motion of player.

The extension of the work can be reconstruction of full human body, may be with a model base approach to avoid the complexity of system. From the animation part the work may be extended to encode and render the animation using other MPEG-4 compression tools for animation i.e. BBA or face and body animation.

Bibliography

[1] 3d presence project. http://www.3dpresence.eu.

[2] Cisco telepresence. http://www.cisco.com/en/US/products/ps7060/index.html.

[3] Collada - digital asset schema release 1.5.0 specification. Technical report, Khronos group, 2008.

[4] Iso/iec 19775-1:2008, x3d (extensible 3d), part 1, architecture and base components, edition 2. Technical report, Web3D Consortium and ISO/IEC JTC1/SC24, 2008.

[5] N. Aspert, D. Santa-Cruz, and T. Ebrahimi. Mesh: Measuring errors between surfaces using the hausdorff distance. In *Proceedings of the IEEE International Conference on Multimedia and Expo*, volume I, pages 705 – 708, 2002. http://mesh.epfl.ch.

[6] Christopher J. Augeri, Dursun A. Bulutoglu, Barry E. Mullins, Rusty O. Baldwin, and Leemon C. Baird, III. An analysis of xml compression efficiency. In *Proceedings of the 2007 workshop on Experimental computer science*, ExpCS '07, New York, NY, USA, 2007. ACM.

[7] Junjie Cao, A. Tagliasacchi, M. Olson, Hao Zhang, and Zhixun Su. Point cloud skeletons via laplacian based contraction. In *Shape Modeling International Conference (SMI), 2010*, pages 187–197, 2010.

[8] Michael Deering. Geometry compression. In *Proceedings of the 22Nd Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '95, pages 13–20, New York, NY, USA, 1995. ACM.

[9] O. Devillers and P.-M. Gandoin. Geometric compression for interactive transmission. In *Visualization 2000. Proceedings*, pages 319–326, 2000.

[10] Faming Gong and Cui Kang. 3d mesh skeleton extraction based on feature

points. In *Computer Engineering and Technology, 2009. ICCET '09. International Conference on*, volume 1, pages 326–329, 2009.

[11] M. Isenburg and P. Alliez. Compressing polygon mesh geometry with parallelogram prediction. In *Visualization, 2002. VIS 2002. IEEE*, pages 141–146, 2002.

[12] Martin Isenburg and Jack Snoeyink Y. Face fixer: Compressing polygon meshes with properties. In *In SIGGRAPH'00 Conference Proceedings*, pages 263–270, 2000.

[13] ISO/IEC. Iso/iec 14496-16:2011, information technology – coding of audio-visual objects – part 16: Animation framework extension (afx). Technical report, ISO, Feb. 2011.

[14] ISO/IEC. Iso/iec 14496-25:2011, information technology – coding of audio-visual objects – part 25: 3d graphics compression model, edition 2. Technical report, ISO, May 2011.

[15] Blagica Jovanova, Marius Preda, and Françoise Preteux. Mpeg-4 part 25: A graphics compression framework for xml-based scene graph formats. *Signal Processing: Image Communication*, 24(1–2):101 – 114, 2009. Special issue on advances in three-dimensional television and video.

[16] Sung-Yeol Kim, J.-H. Cho, A. Koschan, and M.A. Abidi. 3d video generation and service based on a tof depth sensor in mpeg-4 multimedia framework. *Consumer Electronics, IEEE Transactions on*, 56(3):1730–1738, 2010.

[17] Eun-Kyung Lee, Yun-Suk Kang, Yo-Sung Ho, and Young-Kee Jung. 3d video generation using hybrid camera system. In *Proceedings of the 2Nd International Conference on Immersive Telecommunications*, IMMERSCOM '09, pages 5:1–5:6, ICST, Brussels, Belgium, Belgium, 2009. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[18] Seungwook Lee, Bonki Koo, Howon Kim, Changwoo Chu, Byoungjun Kim, Daiyong Kim, KyoungSoo Son, Kiho Choi, Eun-Young Chang, and Euee S. Jang. Quantization-based compact representation of 3d mesh. Technical report, ISO/IEC, 2008.

[19] Khaled Mamou, Titus Zaharia, and Françoise Prêteux. Tfan: A low complexity 3d mesh compression algorithm. *Comput. Animat. Virtual Worlds*, 20(2-3):343–354, June 2009.

[20] Zoltan Csaba Marton, Radu Bogdan Rusu, and Michael Beetz. On Fast Surface Reconstruction Methods for Large and Noisy Datasets. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, Kobe, Japan, May 12-17 2009.

[21] J. Mulligan and K. Daniilidis. Real time trinocular stereo for tele-immersion. In *Image Processing, 2001. Proceedings. 2001 International Conference on*, volume 3, pages 959–962 vol.3, 2001.

[22] Benjamin Petit, Joeffrey Legaux, Jean sébastien Franco, Thomas Dupeux, Bruno Raffin, Ingo Assenmacher, Benoit Bossavit, Emmanuel Melin, and Edmond Boyer. A 3d data intensive tele-immersive grid.

[23] Benjamin Petit, Jean-Denis Lesage, Clément Menier, Jérémie Allard, Jean-Sébastien Franco, Bruno Raffin, Edmond Boyer, and François Faure. Multi-camera real-time 3d modeling for telepresence and remote collaboration. *INTERNATIONAL JOURNAL OF DIGITAL MULTIMEDIA BROADCASTING*, 2010:247108–12, 2009.

[24] M. Preda and F. Preteux. Advanced animation framework for virtual character within the mpeg-4 standard. In *Image Processing. 2002. Proceedings. 2002 International Conference on*, volume 3, pages 509–512 vol.3, 2002.

[25] Jarek Rossignac. Edgebreaker: Connectivity compression for triangle meshes. *IEEE Transactions on Visualization and Computer Graphics*, 5(1):47–61, January 1999.

[26] JarekR. Rossignac. Through the cracks of the solid modeling milestone. In Sabine Coquillart, Wolfgang Straßer, and Peter Stucki, editors, *From Object Modelling to Advanced Visual Communication*, Focus on Computer Graphics, pages 1–75. Springer Berlin Heidelberg, 1994.

[27] Giseok Son, Byeongwook Min, Daiyong Kim, Hyungyu Kim, and E.S. Jang. Simple and fast compression of 3d meshes. In *Convergence Information Technology, 2007. International Conference on*, pages 2175 –2180, nov. 2007.

[28] Gabriel Taubin, André Guéziec, William Horn, and Francis Lazarus. Progressive forest split compression. In *Proceedings of the 25th Annual Conference on Computer Graphics and Interactive Techniques*, SIGGRAPH '98, pages 123–132, New York, NY, USA, 1998. ACM.

[29] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Trans. Graph.*, 17(2):84–115, April 1998.

[30] Gabriel Taubin and Jarek Rossignac. Geometric compression through topological surgery. *ACM Trans. Graph.*, 17(2):84–115, April 1998.

[31] C. Touma and C. Gotsman. Triangle mesh compression. In *In Graphics Interface '98 Conference Proceedings*, pages 26–34, 1998.

[32] R. Vasudevan, G. Kurillo, E. Lobaton, T. Bernardin, O. Kreylos, R. Bajcsy, and K. Nahrstedt. High-quality visualization for geographically distributed 3-d teleimmersive applications. *Multimedia, IEEE Transactions on*, 13(3):573–584, 2011.

[33] Wanmin Wu, Raoul Rivas, Md. Ahsan Arefin, Shu Shi, Renata M. Sheppard, Bach D. Bui, and Klara Nahrstedt. In Wen Gao, Yong Rui, Alan Hanjalic, Changsheng Xu, Eckehard G. Steinbach, Abdulmotaleb El-Saddik, and Michelle X. Zhou, editors, *ACM Multimedia*, pages 877–880. ACM.