

A point-based tele-immersion system: from acquisition to stereoscopic display

Diego Ruiz^a, Benoît Maison^c, Jean-Luc Bruyelle^b, Xavier Desurmont^b and Benoît Macq^a

^a Université catholique de Louvain, Communications and Remote Sensing Laboratory,
Louvain-la-Neuve, Belgium;

^b Multitel ASBL, Parc Initialis, Rue Pierre et Marie Curie 2, B-7000, Mons, Belgium

^cAlterface, 8 chemin des étoiles, B-1348 Louvain-la-Neuve, Belgium

ABSTRACT

We present a point based reconstruction and transmission pipeline for a collaborative tele-immersion system. Two or more users in different locations collaborate with each other in a shared, simulated environment as if they were in the same physical room. Each user perceives point-based models of distant users along with collaborative data like molecule models. Disparity maps, computed by a commercial stereo solution, are filtered and transformed into clouds of 3D points. The clouds are compressed and transmitted over the network to distant users. At the other side the clouds are decompressed and incorporated into the 3D scene. The viewpoint used to display the 3D scene is dependent on the position of the head of the user. Collaborative data is manipulated through natural hand gestures. We analyse the performance of the system in terms of computation time, latency and photo realistic quality of the reconstructed models.

Keywords: tele-immersion, point-based, stereo, stereoscopic, vision-based, hand gestures

1. INTRODUCTION

Tele-Immersion consists of putting two or more partners into contact in such a way that both of them will have the realistic feeling of being physically in front of each other, interacting in a natural way and having both the possibility of handling, in addition, 3D objects or visualizing 3D animations embedded in their virtual environment, or having other advanced possibilities such as interacting with lab tools. Example of applications could be the CAD/CAE design in the automobile industry, making possible a collaborative designing approach between partners distributed in different locations, or the capability of remote handling of instruments and tools in a typical R&D environment.

Photorealistic models, low latency and real time reconstruction improve immersion. Among the possible reconstruction methods, point-based models are often considered. They do not require computing the connectivity, and the sheer number of points allows us to see the model as a whole, like the 2D pixels of an image.

The TIFANIS Project, Tele-Immersion For Applications supporting New Interactive Services, aims at providing a Tele-Immersion application prototype oriented to services. We present a point-based reconstruction and transmission pipeline for a collaborative tele-immersion system. Two users in different locations collaborate with each other in a shared, simulated environment as if they were in the same physical room, in front of each other. Each user perceives point-based models of distant users along with collaborative data like molecule models. Disparity maps, computed by a commercial stereo solution, are filtered and transformed into clouds of 3D points. The clouds are compressed and transmitted over the network to distant users. At the other side the clouds are decompressed and incorporated into the 3D scene. The viewpoint used to display the 3D scene is dependent on the position of the head of the user. Collaborative data is manipulated through natural bare hand gestures. We analyse the performance of the system in terms of computation time, latency and photo realistic quality of the reconstructed models.

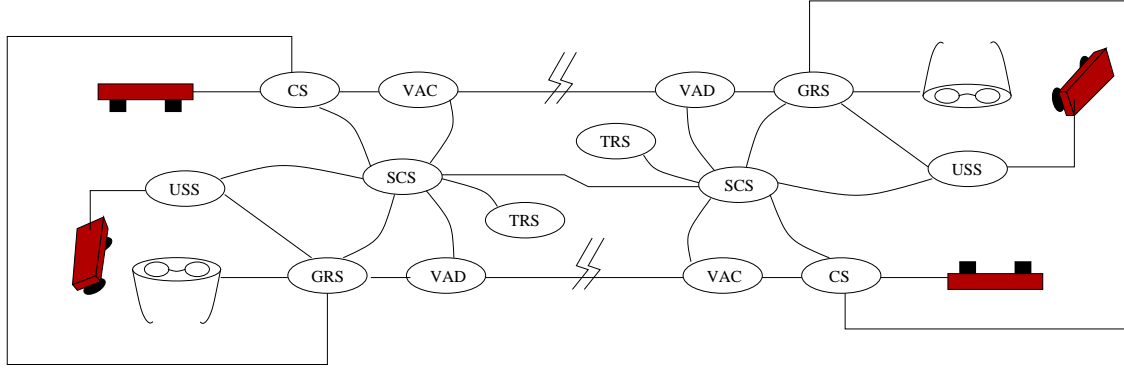


Figure 1. Representation of the whole TIFANIS SYSTEM. CS: Capture Subsystem. VAC: Video and Audio Coding. VAD: Video and Audio Decoding. TRS: Transmission and Reception Subsystem. GRS: Graphical Rendering subsystem. USS: User-System interface Subsystem. SCS: System Control Subsystem.

2. OVERVIEW OF THE TIFANIS SYSTEM

A typical tele-immersion session requires two cubicles, one per user, as represented in figure 1. To describe the system, we will refer to a local cubicle connected through the network to a remote cubicle. This yields the definition of a local user and a remote user. Each user is sitting at a table and perceives a 3D model of the other as seen through a window. For this purpose, the virtual camera used to display the 3D scene is determined by the viewpoint of the user. The displayed 3D scene contains other 2D and 3D objects that the user can manipulate for collaboration. The manipulations performed in one cubicle are relayed to the other cubicle. Two dimension elements like images and text documents are represented as flat windows in the 3D scene and the in-focus 2D element is also displayed on a 2D close-view monitor.

The functionalities of a cubicle are implemented by seven modules responsible for different parts of the system. From the user point of view, the remote Capture Subsystem (CS) acquires clouds of 3D points. Those points are transmitted by the remote Video and Audio Coding subsystem (VAC) to the local Video and Audio Decoding subsystem (VAD). The network transmission, reception and monitoring of Real-time Transport Protocol (RTP) packets is assured by the Transmission and Reception Subsystem (TRS). The VAD passes the decoded clouds to the local Graphical Rendering subsystem (GRS) that displays them in the 3D scene. The remote VAC is also responsible of the audio capture and coding while the local VAD decodes it and renders it. The local User-System interface Subsystem (USS) is the interface by which the user interacts with the system through natural gestures. The local USS communicates to the local GRS how to modify the display of collaborative elements in the 3D scene given the user inputs. The local CS computes the user's viewpoint and transmits it to the local GRS, which adapts the virtual viewpoint used to display the 3D scene. Finally, the System Control Subsystem (SCS) initialises the system, monitors the modules of its cubicle and takes the appropriate recovery action in case of any error.

3. THE CAPTURE SUBSYSTEM

The CS is a computer-intensive subsystem, running on two PCs. A dual-core computer performs the stereo computation, the camera management and the interface with the other subsystems. A second computer, which can be mono processor, runs the detection of the viewpoint. Both PCs use a Gigabit Ethernet interface. A 100 Mbps Ethernet connection would slow down the processing by delaying the transfers between the two PCs running the CS. The output is shown in figure 2.



Figure 2. Validation display of the CS. Top left is the 20K point cloud of the user. Bottom left shows the detected face (green rectangle). Upper right is the depth map. The two small images on bottom and middle right are the results of the parallel computation of the upper and lower parts of the depth map, which are then superimposed to form the complete depth map (upper right).

3.1. Computing a point-based model of the user

The CS uses an integrated stereo camera, the STH-MDCS2-C manufactured by Videre Design.¹ Using a built-in stereo camera allows us a stable calibration in time. The camera is located just in front of the user. The point-based model of the user is built using the depth map calculated from the stereo image provided by the camera. The Small Vision System stereo algorithm* is used for this task. Only the part of the depth map corresponding to the user is of any interest. The background is considered to be everything that is significantly further from the camera than the user, significantly being a fixed threshold set in the CS parameters. The foreground part of the depth map is converted into a cloud of colour 3D points by using the colour image shot by the left camera and by converting the foreground pixels and their depth into 3D coordinates.

3.2. Viewpoint detection

The CS calculates the location in space of the user's viewpoint. The algorithm detects and tracks the face in the left 2D image provided by the Videre camera. Then, it gets its distance to the screen using the depth map provided by SVS³. Extensive testing has shown that it is not necessary to find the eyes. Actually it is much faster, and just as accurate in practice, to use the centre of the detected face, which has been found to lie consistently midway between the eyes, in any orientation of the face.

The viewpoint detection module is composed of two inter-related parts: detection and tracking. The robustness of the module is greatly enhanced by combining these two parts. The system first searches for faces in the whole image, using the algorithm described by Viola and Jones in.^{4,5} Once the face is detected, it is tracked,

*The Small Vision System² (SVS) libraries are provided with the Videre Design stereoscopic camera

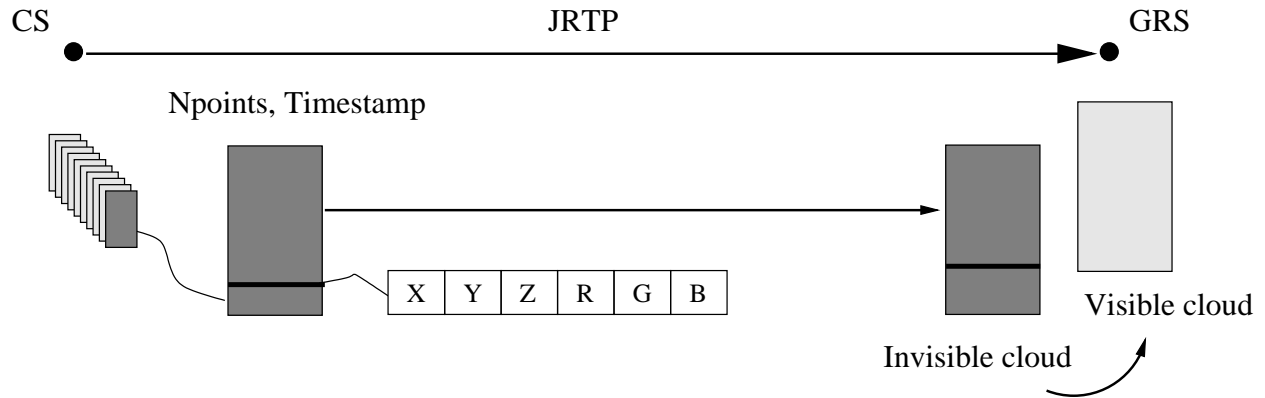


Figure 3. The new cloud produced by the CS has to be transmitted to the invisible cloud of the GRS. There is no buffering of clouds to avoid increased latency.

which allow us to recover faces with such orientations that they would not have been captured by the detection alone. Detection is re-launched periodically, allowing detecting new faces or recovering faces lost by the tracking part.

3.3. Performance

The CS subsystem has been tested as part of the overall validation of the TIFANIS system. It has been found to run seamlessly at 20 fps with 320x240 colour frames, using dual-core, 3.0 GHz, 1 GB RAM. The latency is 50 ms for the calculation of the point cloud and 30 ms for the extraction of the users viewpoint. Some more information of how the real-time requirement is achieved are given in Desurmont.⁶

4. THE CLOUD COMPRESSION AND DECOMPRESSION

As depicted in figure 3, the new cloud computed by the Capture Subsystem should be sent directly to the invisible cloud slot of the Graphical Rendering subsystem. There should be no buffering of clouds in the process. Each cloud is described by its number of points, its timestamp and the points themselves. At the server side (CS), each point is described by three doubles for the X, Y and Z coordinates, and three unsigned chars for the RGB colour. At the reception side (GRS), the points are expected as floats for position and floats between 0 and 1 for colour. We do not require an image structure for the cloud, only a list of points. We do not require preserving the order of points at reception side.

The main constraints are :

- The Transmission and Reception Subsystem requires the Real-time Transport Protocol library JRTPLIB for network monitoring. UDP is required for small latency.
- Global requirements are Small latency, a limited bandwidth of 50 Mbits per second, 20 frames per second and 50 K points per frame.
- We cannot lose the frame if a small number of packets is lost.
- Low CPU usage on the CS and GRS computers.

4.1. Flexible architecture

The real time low latency compression of huge clouds is CPU intensive. For this reason we use a flexible architecture represented in figure 4. If the average number of points is too high, the CPU usage can be transferred from the CS to the compressor computer and from the GRS to the decompressor computer at the cost of increased latency due to the extra TCP local communication.

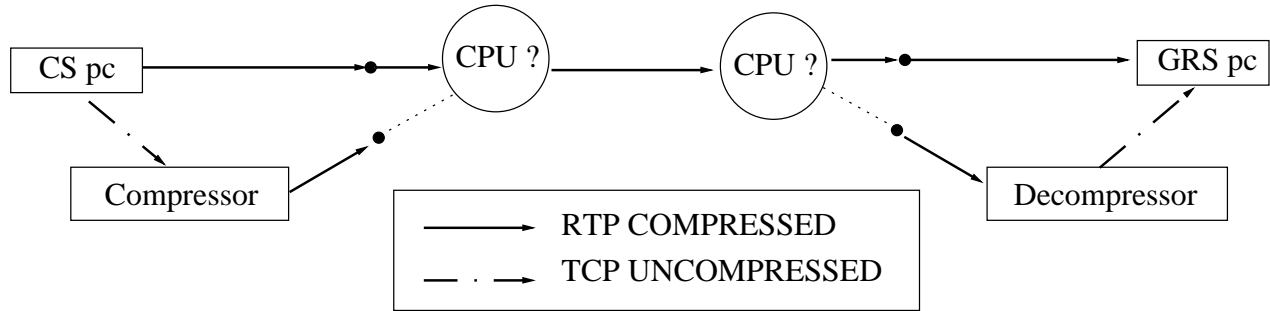


Figure 4. We can trade latency for free CPU on the CS by adding the compressor and on the GRS by adding the decompressor.

4.2. Quantization

For each cloud, we compute the smallest X, Y and Z intervals containing all the corresponding coordinate values of the points. We divide each interval into 2^{16} bins. Each point coordinate is replaced by the id of its interval. There is no colour bit reduction. The bounded area may change from cloud to cloud. The system could benefit from a fixed bounded area at the cost of a small reduction of resolution inasmuch as the bounded area as computed above does not change much from cloud to cloud.

4.3. Huffman compression

Even if the clouds given by the CS do not have an image structure, i.e. there are holes, successive points in a cloud are highly correlated. Figure 5 shows, for each coordinate and each colour component, the histogram of the difference between the value of the point and the value of its predecessor. For each histogram, a small number of bins represents more than ninety percent of the distribution. For example, the Y coordinate is identical sixty percent of the time. Those distributions are good candidates for entropic coding. Furthermore, given the small number of intervals representing most of the distributions, the differences of values are good candidates for Huffman compression. The delta look of the histograms could be increased by a stronger quantization, but we decided to keep a sixteen bits resolution for the coordinates of the clouds. We estimate the probabilities of the bins thanks to the histograms and build the Huffman trees. We remove codes longer than sixteen bits (eight for colour components) by replacing them by the escape code. There is an associated probability with the escape code ensuring that its length is appropriate. The coder and the decoder share the Huffman trees. Our actual system does not re-compute the Huffman trees but it is a planned improvement. The trees have to be computed at the compressor side and sent to the decompressor. They cannot be computed in parallel at the decompressor because some points may be unavailable.

4.4. Per-packet Huffman compression

Our system should be robust to packet loss. A coordinate or colour component is either coded by the Huffman code of the difference with respect to the previous value or by the escape code and the sixteen or eight bits value of the coordinate or colour component. If we perform the Huffman compression by cloud, losing a packet implies losing all the following points. To avoid it, we perform a per-packet Huffman compression. Each packet contains its first point uncompressed and then the list of compressed differences of coordinates. Each packet represents a chunk of successive points in the list and can be decoded in a stand-alone basis. The composition of a packet is represented in figure 6.

4.5. Use Reed Solomon to recover lost packets ?

Reed Solomon allows us to protect k bytes of data with $2t$ parity symbols and to recover t errors.⁷ In order to use Reed Solomon to recover at most t packets lost, we need to distribute the Reed Solomon encoded data

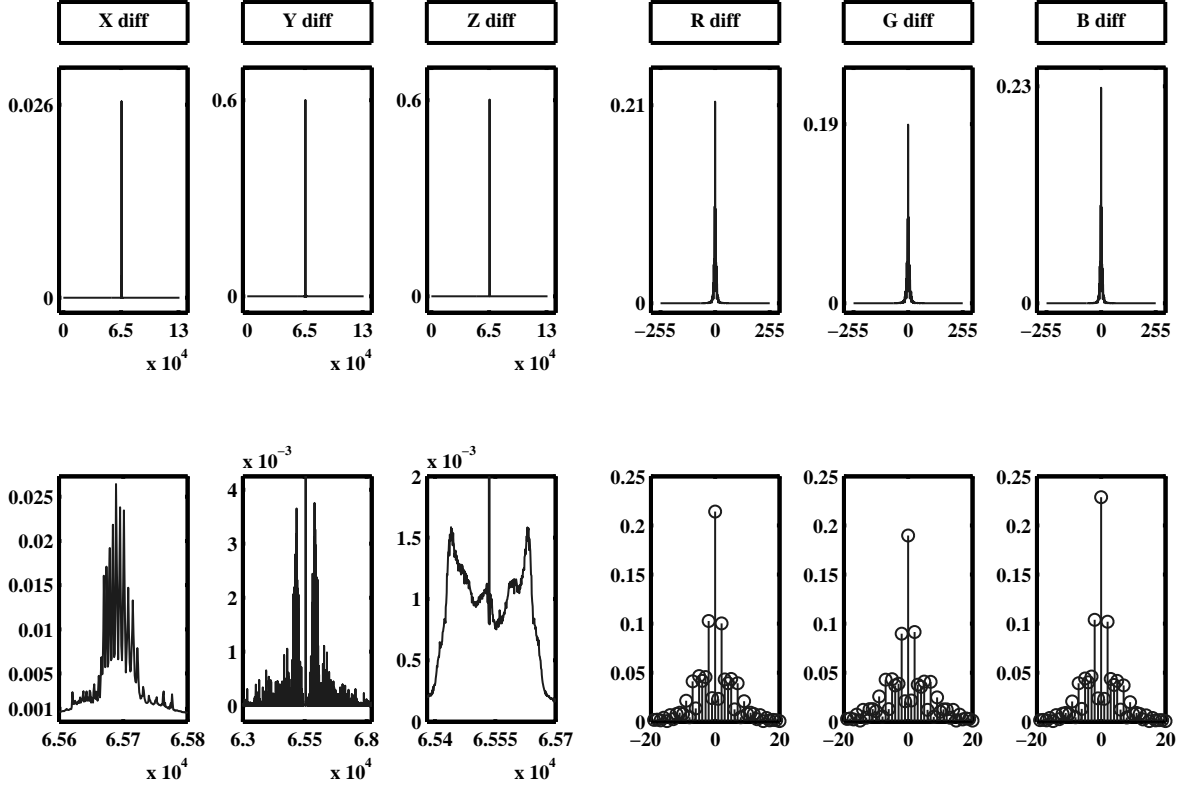


Figure 5. Up: The six histograms corresponding to the X, Y and Z coordinates and R, G and B colour components. Down: A zoom on the important part of the distribution.

Mode	# Points in cloud	Timestamp	Bounding Box	# Points in packet	#Packets in cloud	Compressed data
First point of packet	P2 compressed			Pi compressed		Pn compressed
$HW_X(X_i - X_{i-1})$	$HW_Y(Y_i - Y_{i-1})$	$HW_Z(Z_i - Z_{i-1})$	$HW_R(R_i - R_{i-1})$	$HW_G(G_i - G_{i-1})$	$HW_B(B_i - B_{i-1})$	Pi Escape values

Figure 6. Up: A packet of a Huffman compressed cloud. Middle: zoom on the compressed data. Down: Zoom on the compression of point Pi.

over $k + 2t$ packets, which implies to interleave the compressed data. However, if we lose more than t packets, we will no be able to recover the lost packets. It implies that there will be unknown bytes in the de-interleaved compressed data. In that case, the whole chunk of $k + 2t$ Huffman compressed packets is unusable. Furthermore, t cannot be large with respect to k inasmuch as the Reed Solomon error recovery time increases quickly with t . Finally, Reed Solomon Encoding and decoding increases computation time and latency. We decided not to use Reed Solomon for error recovery.

4.6. Dealing with packet lost or delayed

Due to the UDP transmission, packets do not arrive in order and maybe lost. When receiving a given cloud determined by its timestamp, we may receive packets of previous clouds, of future clouds and of the current cloud. Furthermore, we may not receive some packets of the current cloud. Then we have to decide when to finish

its reception. We take decisions based on the timestamp of incoming packets. Packets of previous clouds are discarded as those clouds have been already rendered or discarded. Packets of the actual cloud are added to the list of packets in whatever order we receive them. If all the packets arrived, the reception of the current cloud is finished. Otherwise the wait time is defined as a number of future clouds. For example, we can set the wait time to three future clouds meaning that we stop the reception of the current cloud as soon as we receive a packet having the timestamp of a fourth future cloud. In the meanwhile, packets of the three first future clouds are buffered. This has a direct impact on latency. When losing a packet of a cloud, the invisible frame will not be filled until the wait time is finished. An inconvenient side effect is that the buffered future clouds may be completed while waiting for the lost packets of the current cloud. Then the invisible frame is swapped quite to quickly. In order to lessen the 3D display acceleration, a timer limits the swapping rate at 40 frames per second. The wait time can be set to zero, meaning that we stop receiving the actual cloud as soon as we receive a packet of a future cloud.

Another artefact arises from the relative order of points in the list. If we lose some successive packets, the decoded cloud will contain horizontal black bands. For example, if we lose the last thirty percent of packets of a cloud, only the upper part of the 3D model of the head of the user will be visible. As a consequence, the user perceives horizontal black bands in some frames and at different height. The height of the band depends on the number of successive packets lost. We lessen the impact of this noise by requiring a percentage of received packets before decompressing or displaying the cloud. If less than ninety percent of packets arrived, we discard the cloud.

4.7. Performance

The transmission of clouds of 50 K quantized points has been tested between two AMD Athlon dual core 3800+ connected through a gigabit Ethernet switch. The server only sends clouds preloaded into memory while the client decompresses them and renders them. The 450 Kbytes quantized clouds are converted into 260 packets of 1 Kbyte. At a frame rate of 20 frames per second, 65 percent of one core of the server computer is used while 90 percent of a core of the client computer is used. The latency is smaller than 100 ms if the wait time is set to zero. The latency of an RTP uncompressed transmission is 40 ms if the wait time is set to zero.

Other tests have been performed at 25 frames per second using the compressor and decompressor computers as shown in figure 4. Each one of the four elements is running on an AMD Athlon dual core 3800+ connected through a gigabit Ethernet switch to the other elements. The TCP uncompressed communication consumes 20 percent of one core of the computer sending the points. At the other side, the TCP reception and rendering of uncompressed points consumes around 50 percent of one core of the GRS computer. In between, the compressor is responsible for the TCP reception of clouds from the server, for their Huffman compression and for their RTP transmission to the decompressor. The compressor consumes 30 and 70 percent of the two cores. The decompressor is responsible for the RTP reception of compressed clouds, for their Huffman decompression and for their TCP transmission to the GRS. It consumes 30 and 60 percent of the two cores. Decoding a Huffman word requires a traversal of the Huffman tree while a simple look up in the Huffman word tables is used at compression side. Thus, we expect the CPU consumption of the Huffman decompression to be higher than the one of the Huffman compression. But we observe a higher CPU consumption at the compressor side. This is due to clouds that are discarded given the packet loss. The server sends an almost constant stream of 11.2 Mbytes per second of uncompressed data to the Compressor while the GRS receives a stream fluctuating between 8 and 12 Mbytes per seconds. The total latency is smaller than 140 ms if the wait time is set to zero.

In the complete TIFANIS system, the capture subsystem only needs to send clouds of around 20 K points at 20 frames per second. It does not consume more than 30 percent of the cores without transmission. To send those clouds, we can directly compress on the capture computer. On the GRS side, a full core is free for decompression. Thus there is no need for compressor and decompressor. Tests showed that 200 Kbytes quantized clouds are compressed into 100 packets of a Kbyte increasing the CPU consumption of the CS to a total usage of 50 percent of both cores. The GRS consumes 25 and 10 percent of its cores. Furthermore, the latency of the RTP compressed transmission is reduced to 50 milliseconds.

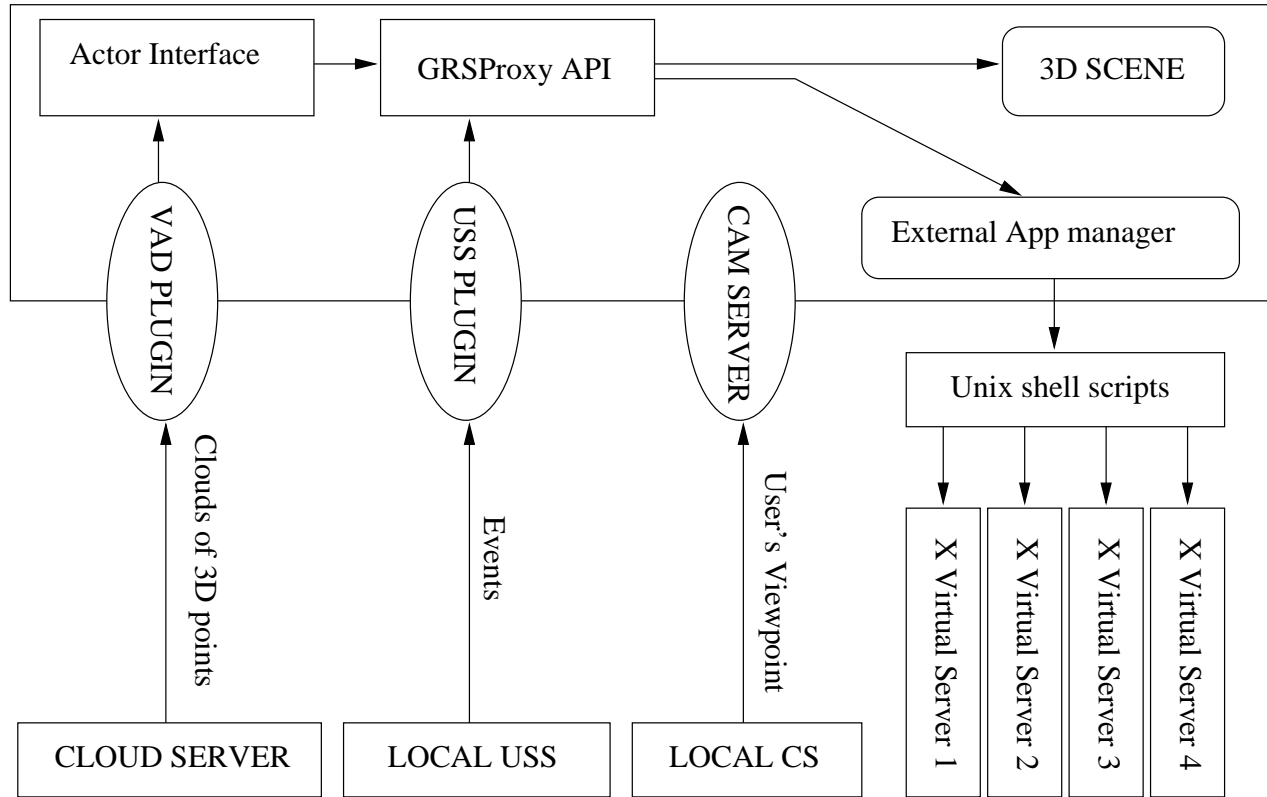


Figure 7. The GRSPROXY API, the synchronization module (not represented) and the camera server form the GRS core. The cloud client and the USS plugin are the external plugins.

5. THE GRAPHICAL RENDERING SUBSYSTEM

User operations on the scene displayed on the GRS of one cubicle have to be replicated in the GRS of other cubicles. Both users in different cubicles have to get the feeling that their changes to the scene are applied to each one of the cubicles in real-time. That implies minimizing latency at replication, avoiding symmetry in the camera point of view, minimizing CPU usage and controlling the synchronization threads. To fulfill those requirements, the GRS has been built as a modular subsystem. There is a core that provides all the rendering and application launching functionalities and a set of external modules that provide the input to this core. This is illustrated in figure 7

5.1. The GRS core

The GRS core is composed of two main layers: the GRSPROXY API below and the synchronization module above.

The synchronization module is responsible of object replication and scene synchronization between the GRS of the connected cubicles. To the API, synchronisation management is transparent. The synchronisation module verifies each of the calls to the GRSPROXY API before actually calling the appropriate method. The synchronisation module is implemented using two threads. If an object has to be replicated, then the first synchronisation thread packs the API call parameters into a message and enqueues it. At the same time, the second thread dequeues the messages and sends them to the remote GRS core via tcp. On the other side, one thread receives the messages and enqueues them, while the other thread dequeues each message and calls the appropriate method with the unpacked parameters. This communication mechanism is bi-directional between two remote GRSs,

allowing for both remote users changing the scene at the same time.

Several modules give shape to the GRSPProxy API functionalities. Some of them are implemented as new nodes in the OpenInventor⁸ tree, like the 3D VNC client, the actor object node used to display the clouds, the regular WRL object importer node or the molecule object node. This last one is an Open Molecular Inventor wrapper plus a PDB file importer. It can display PDB files without consuming the entire resources of the graphics card. That's a key feature, if we consider the number of objects managed on the scene graph and all the modules that handle them. We aim to optimise the CPU and GPU usage of the molecule display to leave enough resources for the other modules in the GRS. Finally, the entry point of external applications are the external application-managing module. This module requires the usage of UNIX shell scripts.

There is an interaction between the CS and the GRS core, the point of view. This feature is implemented using direct communication between them: the CS sends the current frame point of view to the GRS core's camera position server.

5.2. External modules

The first external module is the client-side of the transmission of clouds of 3D points. As previously shown in figure 4, the client may receive point from the local decompressor, from the remote Compressor or from the remote CS depending on the configuration of the TIFANIS system. The client provides the data needed to render the 3D facial point cloud of a remote actor by filling the invisible cloud and swapping the visible and invisible cloud afterwards. The actor-rendering interface shown in figure 7 calls one method that paints the visible cloud in a non-blocking way.

The USS plugin is the second external module of the GRS. It maps the local user interaction events to GRSPProxy API calls. The USS plugin has direct access to all the rendering and application launching functionalities of the GRSPProxy API.

5.3. Rendering

To display all the elements of the 3D scene, including 2D objects, the GRS uses OpenGL functionalities and a dual-head nVIDIA Quadro NVS 280 card with hardware stereo display support. In addition, the in-focus 2D element is displayed on the close view monitor in a second VNC client. The size of the points of the 3D cloud is adjusted during the experimental setup so that the user perceives the model as a whole. It depends on the resolution of the cloud and on the distance between the viewpoint and the 3D Model. If the size is too small, the 3D model is perceived as a collection of points and not as whole 3D object. If the size is too big, we lose resolution. In both cases, we break the photorealistic quality of the models, and thus immersion.

6. THE IMMERSIVE USER INTERFACE

The promise of the immersive interface is to give the users a realistic feeling of being there in person, by seeing each other as if they were in the same room, and by interacting with virtual objects as they would with real objects. It was decided early on that unencumbered interaction was desirable, because requiring the user to wear motion sensing equipment, or use special input devices, would deter from the naturalness of the interface and of the tele-presence experience. We made an exception for stereo-viewing glasses, because they can be relatively unobtrusive, and because they currently are the only cost-effective 3D viewing solution.

However, in order to deal with the current limitations of image acquisition, signal processing, and display technologies, we need to strike a balance between those goals and the general usability of the system.

Hence, the research challenge presented by the immersive user interface is two-fold:

1. offer the user intuitive and efficient means of interaction with the 3D virtual workspace and the system, given the limitations of 2

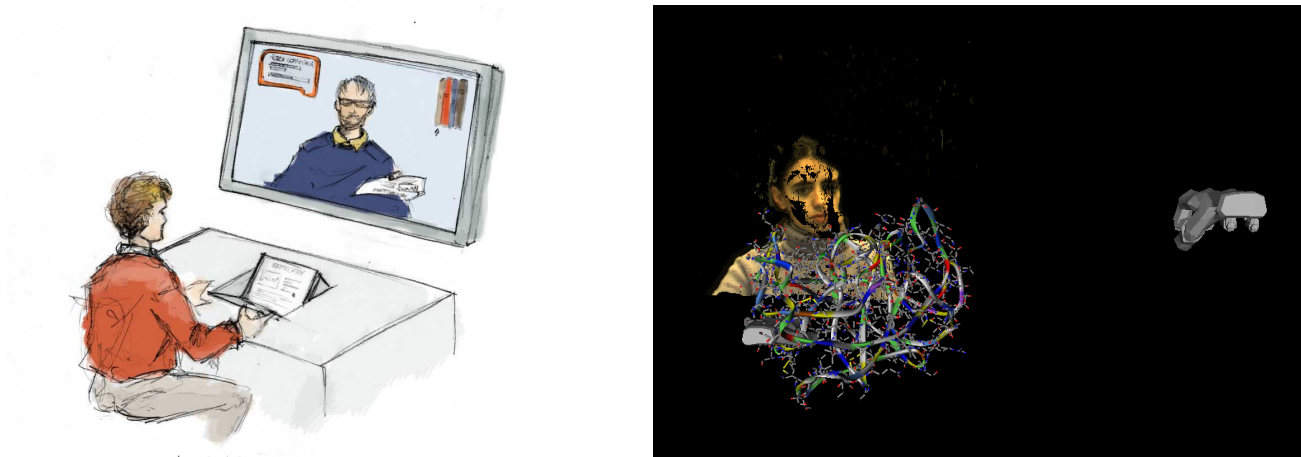


Figure 8. left: close-view monitor and main display, right: 2D view of workspace with remote user, hand pointers and molecular model

2. develop robust image processing algorithms for gesture recognition in order to make 1 possible.

6.1. A mixed 2D/3D user interface

The close-view monitor

The objects shown in the 3D shared virtual workspace are not limited to 3-dimensional objects, but include 2-dimensional ones like text, spreadsheets, graphics, or the user interface of external applications. In order to preserve the immersive experience, it is necessary to display 2D objects as windows in 3D space. However, unless such a 2D object is made so large as to take up a large fraction of the screen real estate, working with it for any extended period may prove uncomfortable to the users. We came up with a hybrid solution to this problem, by adding a smaller conventional display, dubbed the close-view monitor (see Figure 8-left), in a recess in the cubicle desk, where 2-dimensional objects are mirrored. The mirroring happens automatically and is updated continuously: whichever 2D object has been touched last is shown live on both users' close-view monitor *in addition* to the shared 3D workspace. The users still have the option of examining the object in the 3D workspace at the scale they wish, but they have at any time the convenience of watching the same object (in its current state) on their personal monitor. Additionally, since a keyboard and a mouse are provided to interact with the objects that require it, any ambiguity about which object is receiving the mouse and keyboard input is resolved at the same time: their input is simply forwarded to the object that is mirrored on the close-view monitor. As a result, the interaction with conventional 2D mouse and keyboard applications is as efficient as in a classical desktop environment, and the 3D immersion metaphor is preserved.

In addition to its main purpose of showing shared 2D documents, the close-view monitor also serves to display the user controls of the tele-immersion system, namely the login/logout dialogs, the connection dialogs, the object selection windows, and the sound volume control.

Hand gestures for 3D object manipulation

Gesture-based interaction is used where it matters most: for the manipulation of 3-dimensional object in the shared workspace. The user can grab with one or two hands any object present in the 3D scene, and move it, or grab it with two hands, rotate it, and scale it as they would rotate or stretch a physical object. To give visual feedback to the users, hand-shaped pointers are inserted in the 3D scene, along with the objects they can manipulate, and the point-based model of the remote user (see Figure 8-right).

6.2. Real-time vision-based gesture recognition

The user's hands are tracked by means of a Stereo-on-a-chip Videre Design¹ stereo camera, distinct from the CS camera, without any additional markers or equipment. The camera is located above and in front of the user, who is allowed to freely move both hands in the air. Two serious difficulties arise from this setup: the hand pose relative to the camera is not constant, and the image of the hands is not captured against a fixed background, but sometimes against the users's desk, the floor, or the user's body. More controlled setups, for example capturing the hand against a flat surface,⁹ while achieving impressive results, do not suffer from either of those problems.

With that in mind, we designed our image processing algorithms with the following objectives:

- robust tracking of both hands,
- resistance to noise and defects in the depth information supplied by the stereo camera,
- real-time operation,
- detection of hand opening and closing.

This last feature is the basis of all the gestures available to the user.

The algorithms we designed operate directly on tri-dimensional data provided by the stereo camera. The position of the camera in the cubicle is determined through a one-off calibration procedure. After that, the camera image and depth information can be converted to a 3D cloud of points. An adjustable volume of interest serves to select the points that are expected correspond to the user, and reject floor, walls or furniture. The user point cloud is analyzed to find the chest, head, and arms of the user, provided the latter are extended. Special care is taken to overcome defects in the depth maps such as missing or incorrect values. In a first pass, the location of the hands is estimated to be at the tip of the arms. A temporal tracking procedure is then used to refine the position of the hands, as well as remove the jitter caused by detection errors and the temporal variability of the depth map provided by the stereo camera. Once the position of the users hands is detected, the next procedure detects the opening and closing motions of the hands. Detection of the skin colour is the first and most important step towards locating the contours of the hand with precision. Other features include the variation in estimated arm length (including the hand), and the visual appearance of the hand. Finally, a temporal filter examines the output of the classifiers for successive frames, and determines when the hands open or close.

7. CONCLUSION

Using five desktop computers, two stereoscopic cameras, a dual-head nVIDIA Quadro NVS 280 and a pair of stereo-viewing glasses per cubicle, each user perceives a 20K point-based model of a distant user along with collaborative data like molecule models. The point based model is displayed at twenty frames per second and the total latency is around one hundred miliseconds[†]. Experimental evaluations show that when a few packets of a frame are lost, the available packets are well decoded and the user does not notice the missing points due to the frame rate. If too much packets are missing due to the UDP transmission, the cloud is discarded. The consequent frame rate reduction depends on the quality of the network. The immersive experience, as well as the breadth of applications of the system, are reinforced by the direct manipulation of shared objects through bare hand gestures.

8. ACKNOWLEDGEMENT

This work has been funded by the Walloon Region within the scope of CELTIC TIFANIS project.

[†]Without latency due to the transmission of RTP packets through the network. The total latency has been measured by sending points to the local GRS instead of the remote GRS.

REFERENCES

1. <http://www.videredesign.com>.
2. <http://www.ai.sri.com/~konolige/svs/svs.htm>.
3. X. Desurmont, I. Martinez-Ponte, J. Meessen, and J.-f. Delaigle, “Nonintrusive viewpoint tracking for 3D for perception in smart video conference,” in *Three-Dimensional Image Capture and Applications VII. Edited by Corner, Brian D.; Li, Peng; Tocheri, Matthew. Proceedings of the SPIE, Volume 6056, pp. 106-117 (2006).*, B. D. Corner, P. Li, and M. Tocheri, eds., pp. 106–117, Feb. 2006.
4. P. Viola and M. Jones, “Rapid object detection using a boosted cascade of simple features,” 2001.
5. R. Lienhart and J. Maydt, “An extended set of haar-like features for rapid object detection,” in *Proceedings of the International Conference on Image Processing*, pp. 900–903, IEEE, (Rochester, USA), September 2002.
6. X. Desurmont, J. Bruyelle, D. Ruiz, J. Meessen, and B. Macq, “Real-Time 3D Video Conference On Generic Hardware,” in *Real-Time Image Processing 2007, part of the IS&T/SPIE Symposium on Electronic Imaging 2007, 28-30 January 2007 in San Jose, CA USA*, 2007.
7. S. Lin and D. J. Costello, *Error Control Coding, Second Edition*, Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.
8. J. Wernecke, *The Inventor Mentor: Programming Object-Oriented 3D Graphics with Open Inventor, Release 2*, ADDISON-WESLEY, Boston, February 2003.
9. C. von Hardenberg and F. Bérard, “Bare-hand human-computer interaction,” in *Proceedings of the 2001 Workshop on Perceptive User Interfaces*,