VTK Remote Rendering of 3D Laser Scanner Ply files for Android Mobile Devices

Release 1.00

Fabio De Pascalis

May 5, 2014

ENEA, CR Brindisi (Italy), fabio.depascalis@enea.it

Abstract

Three-dimensional laser scanner is a modern technology used to detect and capture the shape of complex objects. There are a lot of possible application fields for this technology: applications in the industrial sector (reverse engineering), in the medical field or in the cultural heritage field for virtual reconstruction of works of art. In this work we have used a new prototype of laser scanner RGB-ITR (Red Green Blue Imaging Topological Radar). It's an amplitude-modulated, 3D colour laser scanner entirely designed at ENEA laboratories[3]. This instrument incorporates three laser beams (450nm, 532nm, 650nm) and can simultaneously record range and self-registered RGB colour data. The main goal of this work is to make accessible the three-dimensional models obtained from laser scans on mobile devices (smartphones, tablets), exploiting a remote rendering application on high-performance workstations[5, 1, 2, 4]. This paper also shows how this can be done fairly easily using one of the most popular instruments in the field of scientific visualization, VTK. The rendering performances of the new mobile devices are increasing dramatically with the use of ad hoc graphics cards (NVIDIA, Adreno). However, modern multicore graphics workstations have much higher performances with respect to mobile devices, and they still remain the only instruments able to visualize polygonal datasets larger than a certain threshold. The solution proposed in this paper is based on a high performance VTK server to render 3D datasets and an android client that manages the interaction with the 3D scene and receives in real-time image streaming from the server (fig.1). In the next sections we give a desciption of the entire system architecture focusing on a particular implementation of a vtk polygonal mapper, `vtkCustomPlyMapper`, (go to journal website to view code and examples) that we have customized in order to optimize and to improve the rendering velocity of files produced by our laser scanner device.

Contents

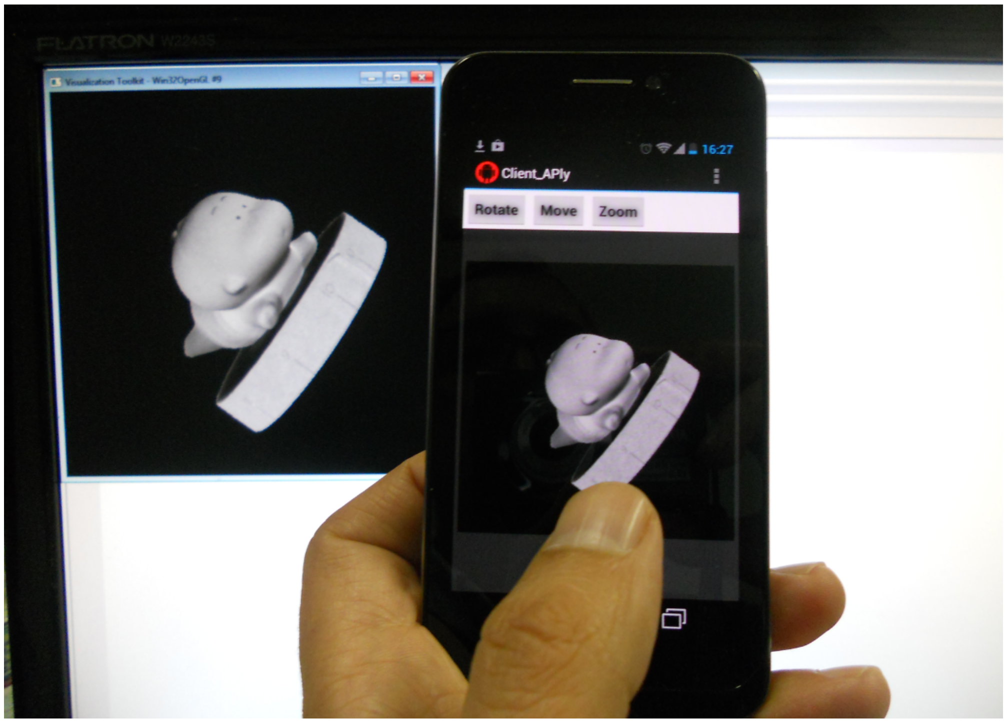

Figure 1: Remote rendering system running on the server (background) and the android client (foreground)

1 Polygonal File Format

The ply file (polygonal file format) is a format developed at the University of Stanford. It's particularly suitable for representing polygonal objects and three dimensional data produced by 3D laser scanners. It consists of a header followed by a list of vertex and a list of polygons. The header specifies the number of vertices and the number of polygons that make up the 3D model and the properties (such as coordinates, colours) associate to each of them. The polygons are defined specifying a number ”n” which identifies the number of sides and specifying a list of indices that represent pointers in the list of vertex. Our laser device generates 3D models composed only from triangular cells (three floating point vertex), and coordinates rgb colour coding (unsigned char) are linked to each vertex.
Here a classic example of our ply format file header:

```
ply
format binary_big_endian 1.0
comment:  XXXXXX
element vertex 847061
property float x
property float y
property float z
property uchar red
property uchar green
property uchar blue
element face 1678492
property list uchar int vertex_indices
end_header
```

2 System Architecture

The proposed system is based on a client/server architecture consisting of an android client for mobile device that communicates with a 3D workstation (server) through wireless connection. The server comprises different components linked each other (fig.2). Event Manager component is used to manage the comunication with the client and to interprete clients events and commands. VTK Rendering module updates the 3D model transformation matrix according to the Event Manager's output and carries out the rendering of the scene. Encoder component extracts from the frame buffer the image resulting from the rendering process, compresses it in jpeg file format and then sends the image to the client over TCP channel. The client has been implemented using Java and Android library. Like the server, it's composed from different subsystems. Event Generator is used to manage and to send a series of commands to the server. It can send messages to require the list of available ply files (stored into the server), to specify the selected file and to ask the server the rendering of the same. It also generates all the events for the 3D models interaction management. Decoder module decompresses the received image and the Viewer component displays it like a bitmap in an android graphical window. The server remains in a permanent waiting state (idle loop) listening for any connection requests. Once a connection is accepted, it's handled in a separate thread setting a transmission remote channel (remote socket). This allows simultaneous management of independent (multithreading) rendering processes for a number of clients. The next section describes in more detail some aspects related to the software implementation and to the involved libraries.

3 Software implementation

3.1 VTK Server

The server has been implemented in C++; its rendering engine has been developed using VTK graphics library while the graphical interface has been realized using WxWidgets library. VTK stands for "The Visualization Toolkit". The choice of this library is due to its capacity in terms of visualization and to the possibility of introducing new basic functionality acting directly on the source code distributed as open source, furthermore it allows to deploy cross platform applications. VTK includes a set of classes written in C++ and supports the OpenGL graphics standard. Let's see an excerpt of the C++ code that puts in evidence as using VTK classes we are able, with few instructions, to get the display of 3D objects in ply format:

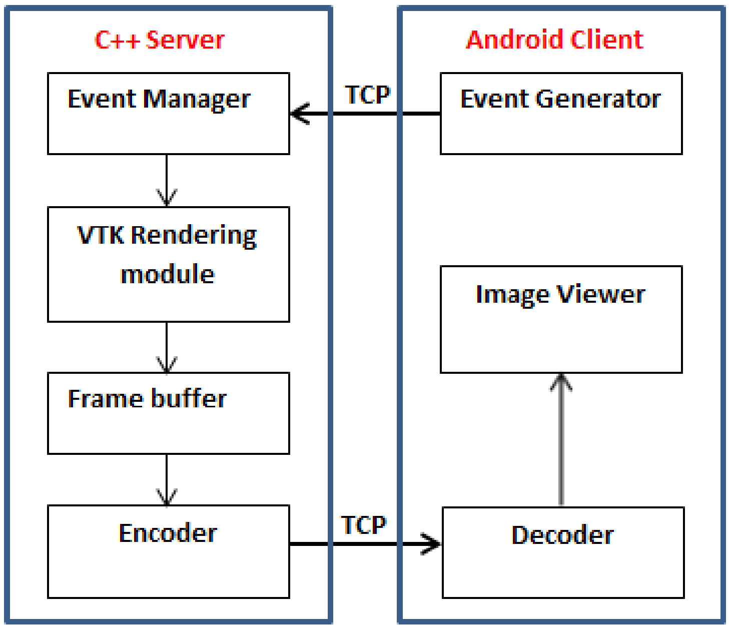

Figure 2: Client/Server system architecture

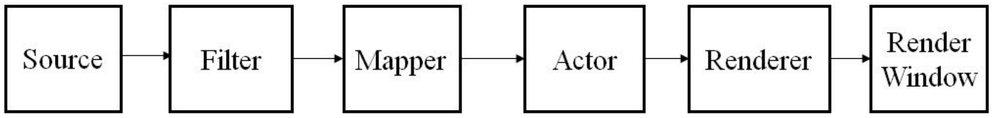

Figure 3: VTK Pipeline.

```
vtkPlyReader *reader = vtkPLYReader::New()
reader-> SetFileName(file);
vtkOpenGLPolyDataMapper *mapPD = vtkOpenGLPolyDataMapper::New();
mapPD->SetInput(reader->GetOutput());
vtkActor *actorPD = vtkActor::New();
actorPD->SetMapper(mapPD);
vtkRenderWindow *renWin = vtkRenderWindow::New();
vtkRenderer *renderer = vtkRenderer::New();
renderer->AddActor(actorPD);
renWin->AddRenderer(renderer);
renWin->Render();
```

In practice, the VTK display process is realized setting up a graphics pipeline linking a number of VTK objects (fig.3)

`vtkOpenGLPolyDataMapper` is a standard VTK subclass derived from `vtkMapper` and is generally used to view and generate OpenGL graphics primitives for data with any kind of polygonal topology, however its generality penalizes the rendering performance. For this reason we decided to replace it with a class, `vtkCustomPlyMapper`, derived from it, specifically developed for viewing our PLY files whose cells are composed only from triangular elements. Basically, the change was done overriding some base class methods, including the methods `DrawTStrips`, `DrawPoints` and `DrawPolygons`, in which we have implemented

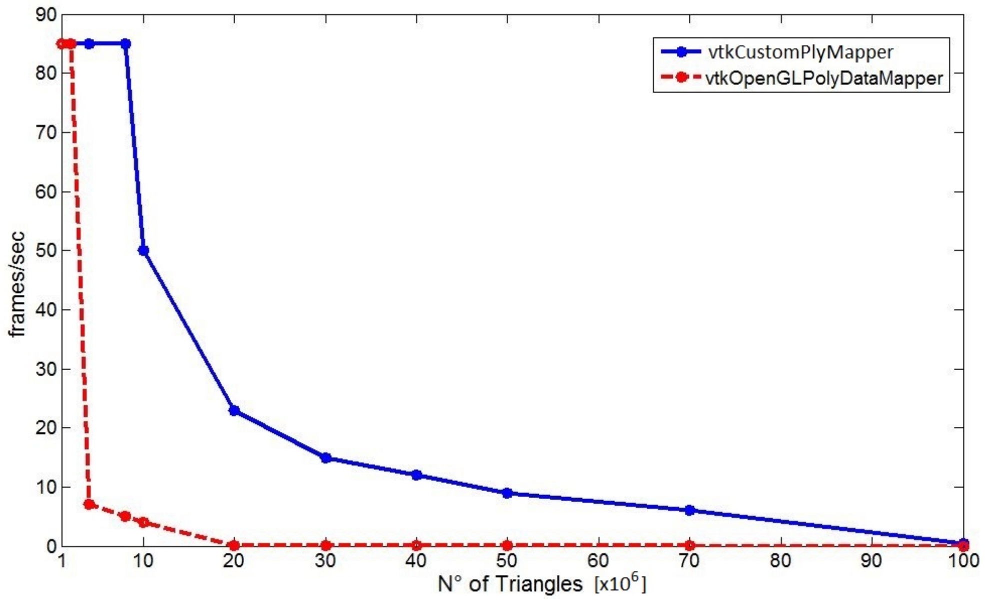

Figure 4: Comparison of frame rate measured using the base class (vtkOpenGLPolyDataMapper) and the derived class (vtkCustomPlyMapper), in the x-axis is shown the number of models triangles, while in the y-axis is shown the number of frames per second.

`glDrawElements` function that allows to specify multiple geometric primitives with very few subroutine calls, instead of calling on a GL function to pass each individual vertex, normal or colour. This makes it possible to substantially increase the frame rate as shown in the graph in figure 4, where we compare the performance of two previous classes (the tests were performed on a Intel core i3 S30 processor, with 16Gb of RAM and a NVIDIA QUADRO 4000 GPU, Windows 7 64bit). Clearly this performance improvement is crucial for a remote rendering service with an efficient level of interactivity. Our laser device associates to each vertex a colour coding that arises from the effects of transmission and reflection of light and, therefore, inherently incorporates shading information. We decided therefore to disable as default the OpenGL lighting in our mapper. In any case, if you need (for example with ply files without colour property) you can add in the pipeline the vtkPolyDataNormals module (to calculate the normals for each vertex) and enable the lighting using `SetLightingOn()` method. In conclusion, although in this first release `vtkCustomPlyMapper` has some limitations, from being adapted to the files produced by our scanner (just 3d models with triangular cells, and properties related only to points and not to cells), it allows us to obtain a much higher rendering speed than `vtkOpenGLPolyDataMapper` on the server, and hence a fluent navigation on the mobile device.

3.2 Android Client

In order to develop the client we have decided to use mobile devices with Android operating system. The architecture of Android is very sophisticated and, despite some inevitable imperfection due to the young age of the platform, it is a modern, reliable, comprehensive and above all expandable system. A further strength of this system is that the Google's platform is completely open-source. Using an integrated environment like Eclipse, Android developer tools and java SDK, we have implemented a client application composed of three android activities (fig. 5). The first activity takes care of the issue involved in a socket instantiation for TCP Client/Server channel, the second activity presents to the user a list of available ply files, while the third activity manages the rendering process and the user interaction (rotate/pan/zoom) with 3d model.

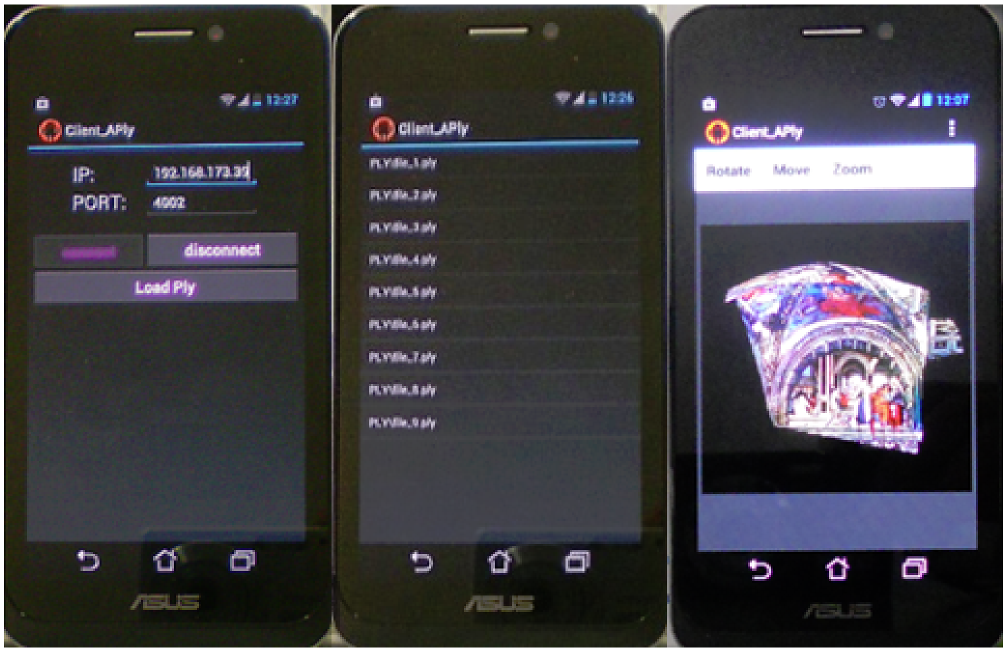

Figure 5: The three Android activities

The exchange of messages between client and server is realized through a communication protocol based on short ASCII commands. For example the interaction with the three-dimensional model is made translating multi-touch user's events on the device touch screen in packets of strings describing the coordinates of the path traced by the user's hand. The server determines, parsing the string, the transformation matrix to apply to the 3D model, then it renders the scene, compresses the image and streams it to the client, which in turn decompresses and displays it as a bitmap on its interface. In such a way the user navigates the scene transmitting inputs and events to the server application by means of android client, but storage and computation resources are provided by VTK server system. From the client's GUI is also possible to change rendering settings such as to enable or not the shading or to set anaglyph stereoscopic 3D effect.

4 Results

Tests have been performed using the following hardware systems:

SMARTPHONE ASUS Padfone, 1GB of RAM and Android 4.0 operating system
WORKSTATION Intel Core i3 S30, 16Gb of RAM, Nvidia Quadro 4000 GPU, Windows 7 64bit

The android client can establish a connection to the C++ Server through a Gigabit Ethernet LAN and a Wi-Fi 802.11n access point equipped with four Ethernet ports (wired speed of up to 100 Mbps) or through a 3G internet connection. Our system has been evaluated with datasets of different dimensions measuring the rendering speed (number of frames per second). In this test VTK rendering context size has been set to 512x512 (RGBA) pixels, therefore the rendered image size is one Mbytes. To limit the transmission time, before being sent to the server the image is compressed in jpeg format using a quality factor of 75. Table 1 shows the frame rates obtained on server side as well as on client side using both WI-FI and 3G connections. The bandwidth in download, measured using wireless link is around 40 Mbps

Table 1: Server and Client frame rate [frames/sec]

Model's Triagles	Server fps	Client fps	
		IEEE 802.11n	3G
10^6	85	25	7
$2*10^6$	85	25	7
$4*10^6$	85	25	7
$8*10^6$	85	25	7
$10*10^6$	50	22	6.6
$20*10^6$	23	14	6
$30*10^6$	15	10	5
$40*10^6$	12	8	4.5
$50*10^6$	9	7	4
$70*10^6$	6	5	3.3
$100*10^6$	0.4	0.38	0.36

while over 3G connection is about 4 Mbps. As expected, the values (tab. 1) reported in columns three and four are lower than those shown in column two, as the client's frame rate is also affected by the delays related to the client events management, image compression ($\approx$ 10 ms), image streaming ($\approx$ 2 ms in WLAN environment and 20 ms in 3G), as well as to the decompression and repainting of the bitmap image in the android interface ($\approx$ 10 ms). Unlike the other delays the rendering time is not constant but increases with the size of the model. In order to achieve maximum interaction rate, also for larger dataset, the application has been provided with a functionality (activated from the client) that reduce the object size through decimation during the interaction phases. The proposed solution enables to have good values in frame rate even for huge size datasets, the jpeg compression algorithm is useful to limit the bandwidth for the image streaming and at the same time the use of OpenGL vertex array allows the server to finish the remote rendering of 3D models in a few msec. With a proper quality connection we can view and interact with a laser scanner dataset without having to make the file downloading.

5 Conclusions

In this paper we describe a method to perform an interactive display on mobile devices of files (ply) generated by 3D laser scanners. Modern smartphones and tablets are provided with graphics hardware acceleration support, however when the size of the dataset exceeds a certain value the characteristics and resources of such devices are not sufficient to perform the rendering, one alternative approach in such cases is the remote rendering.
In particular we have developed an application, using only open source instruments (WxWidgets, Android SDK, VTK), in which the computation and rendering process are delegated to a performing remote system, and the handled device is only used for viewing and managing the user interaction with the 3D dataset. This solution offers the advantage of being able to treat with large size 3D models, moreover it overcomes the necessity to transmit and process the source files locally on the user's device.

References

[1] C.F. Chang and S.H. Ger. Enhancing 3d graphics on mobile devices by image-based rendering. *Proceeding IEEE Third Pacific Rim Conf. Multimedia*, pages 1105 – 1111, 2002. (document)

[2] Joachim Diepstraten, Martin Gorke and Thomas Ertl. Remote line rendering for mobile devices. *Proceedings of Computer Graphics International (CG104)*, pages 454 – 461, 2004. (document)

[3] R. Ricci, L. De Dominicis, M.F. De Collibus, G. Fornetti, M .Guarneri, M. Nuvoli and M. Francucci. RGB-ITR: An amplitude-modulated 3d colur laser scanner for cultural heritage applications. *Lasers in the Conservation of Artworks VIII*, pages 191 – 197, 2011. (document)

[4] F. Lamberti and A. Sanna. A streaming-based solution for remote visualization of 3d graphics on mobile devices. *IEEE Transactions on Visualization and Computer Graphics Vol. 13, N. 2*, pages 247 – 260, 2007. (document)

[5] L. McMillan. An image-based approach to three-dimensional computer graphics. *Ph.D. Dissertation. Technical Report,University of North Carolina at Chapel Hill*, pages 97–103, 1997. (document)