

3D Post Processing Methods for Web Based Integration

RODRIGUES, Marcos

Available from Sheffield Hallam University Research Archive (SHURA) at:

<http://shura.shu.ac.uk/5281/>

This document is the author deposited version. You are advised to consult the publisher's version if you wish to cite from it.

Published version

RODRIGUES, Marcos (2009). 3D Post Processing Methods for Web Based Integration. In: Brazilian National Conference on Hypermedia Environments for Learning., Florianopolis, Brazil, 4-9 Nov 2009. (Unpublished)

Repository use policy

Copyright © and Moral Rights for the papers on this site are retained by the individual authors and/or other copyright owners. Users may download and/or print one copy of any article(s) in SHURA to facilitate their private study or for non-commercial research. You may not engage in further distribution of the material or use it for any profit-making activities or any commercial gain.



Marcos A Rodrigues

Geometric Modelling and Pattern Recognition Research Group - GMPR
Sheffield Hallam University, Sheffield, UK

m.rodrigues@shu.ac.uk

www.shu.ac.uk/gmpr

3D POST PROCESSING METHODS FOR WEB BASED INTEGRATION

Motivation for 3D Imaging

3D Imaging can provide:

- Realistic models
- Interactive visualization and manipulation of objects on screen
- Sense of immersion in a 3D world
- Unique opportunity to explore objects and relationships
- Control over how objects are displayed
- A variety of interrogation dialogues as opposed to a fixed script
- Better recall of learning experience from a visual and spatial environment

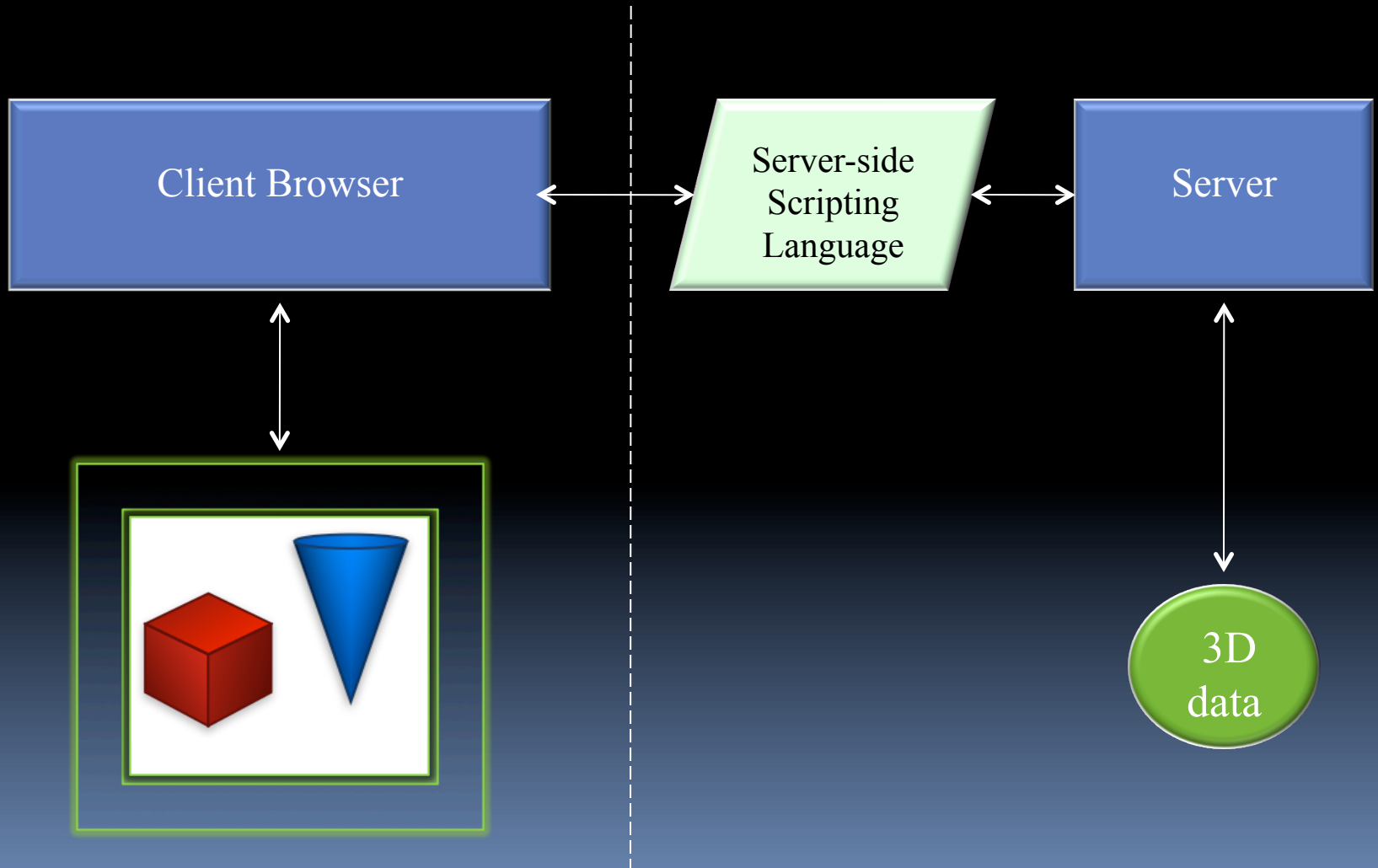
These lead to enhanced learning experiences

Some Limitations of 3D Imaging

- Web browsers do not understand 3D models
- Current methods are too slow to be used in a realistic scenario within a web browser
- Problems of data representation, compression, processing time
- Need a number of techniques to generate 3D models

Our needs are for real-time, responsive interaction with 3D scenes within a browser

Existing Option

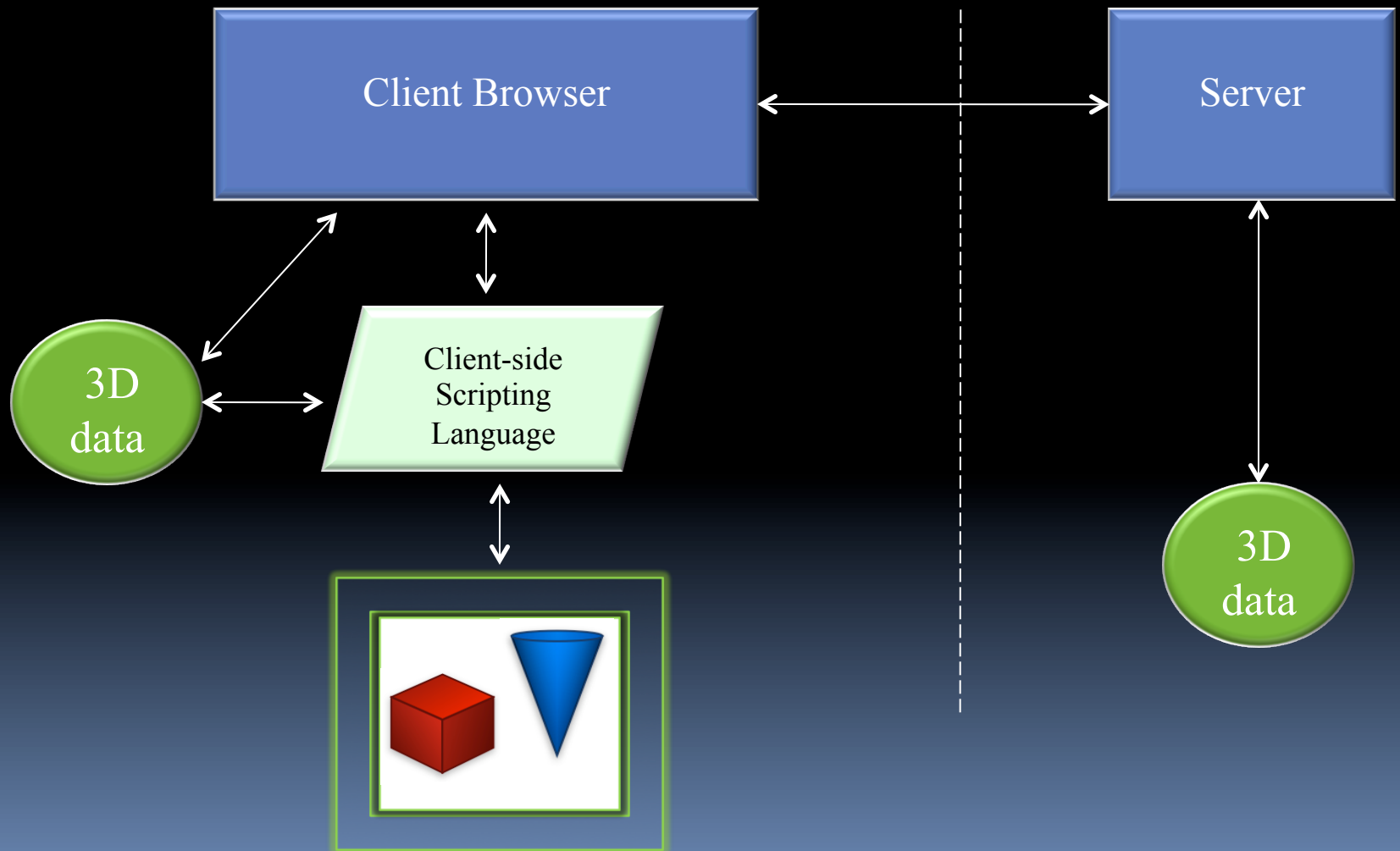


Some Problems with this Approach

- If the scene is to be rotated or translated the server has to calculate new pixel values for the displayed image
- Too much overhead communication between client/server
- The burden is mostly on the server, uneven load (fat server and thin client)
- Works well for 2D images, streaming audio and video, and text data from databases

A workable solution for interactive 3D would be to place the load on the client machine taking advantage of the underlying hardware thus reducing overhead communication

A Workable Solution



The Obvious Options (1)

JavaScript

- Client-side scripting language
- Developed to add dynamic content to web pages
- JavaScript statements are embedded into html code
- Compatible with all browsers
- The most popular scripting language on the Internet

Thus, potential for easy integration of 3D contents into a web page

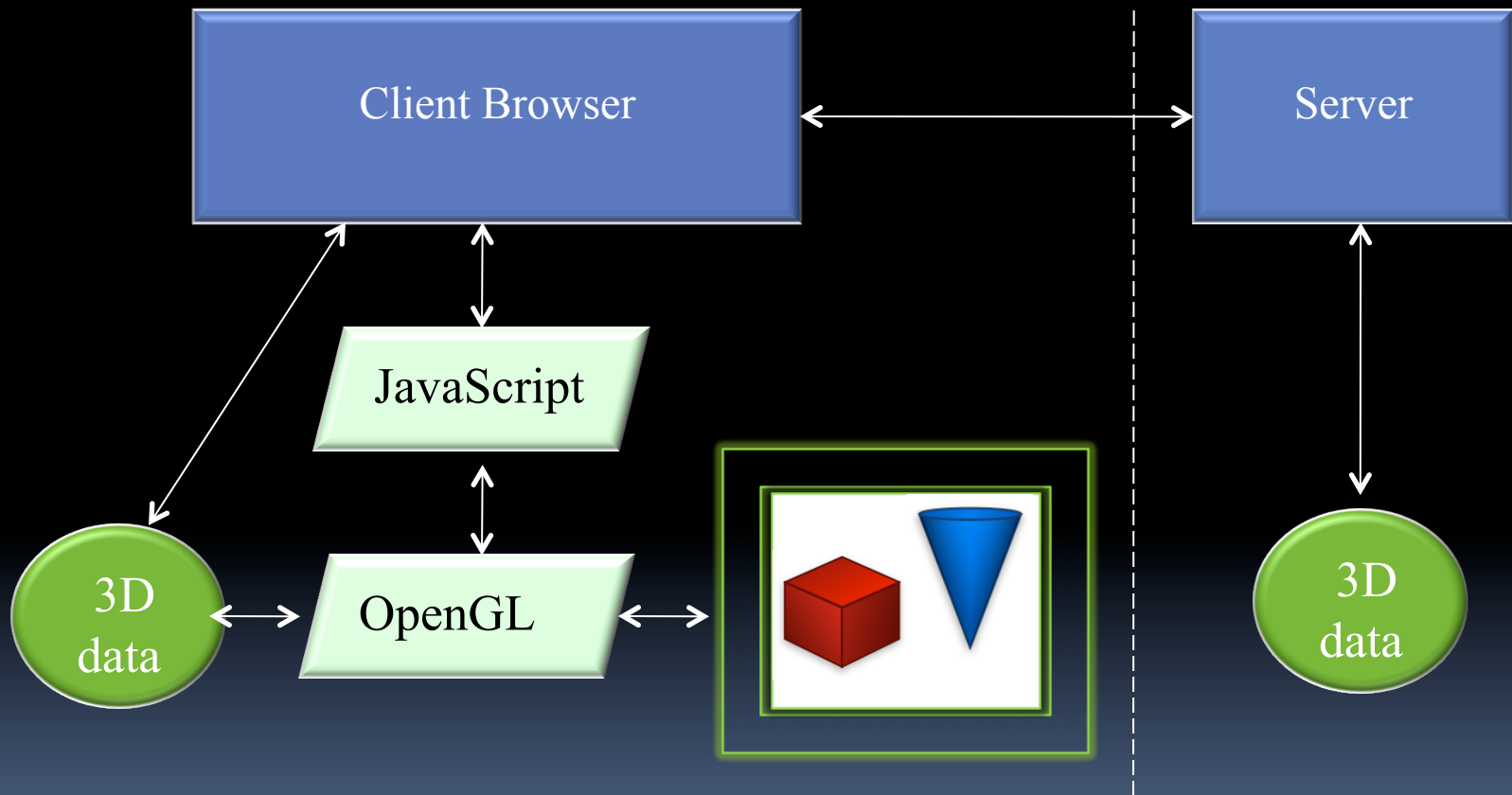
The Obvious Options (2)

OpenGL

- The industry standard for high performance graphics
- Comes pre-installed on all major operating systems (Windows, Mac, Linux, Unix)
- Runs on the client machine taking advantage of graphics hardware
- Developed in C/C++ so it requires some programming knowledge to be able to drive the OpenGL API
- Vast number of tutorials on the web

However, since we just want to include interactive 3D models on the web we may not want to learn OpenGL at all !

THE Solution



The problem at this juncture is that JavaScript does not understand OpenGL

The Mozilla Approach

- The future of the web is augmenting 2D content with the 3rd dimension
- Khronos: a 3D web project seeking to create standards for 3D solutions development
- The aim is for end users to take advantage of those solutions as part of the normal web experience
- Standards are expected to be published by May 2010
- The approach
 - To develop an OpenGL wrapper within JavaScript
 - Effectively, a JavaScript extension
 - The browser will come pre-installed with all necessary plug-ins

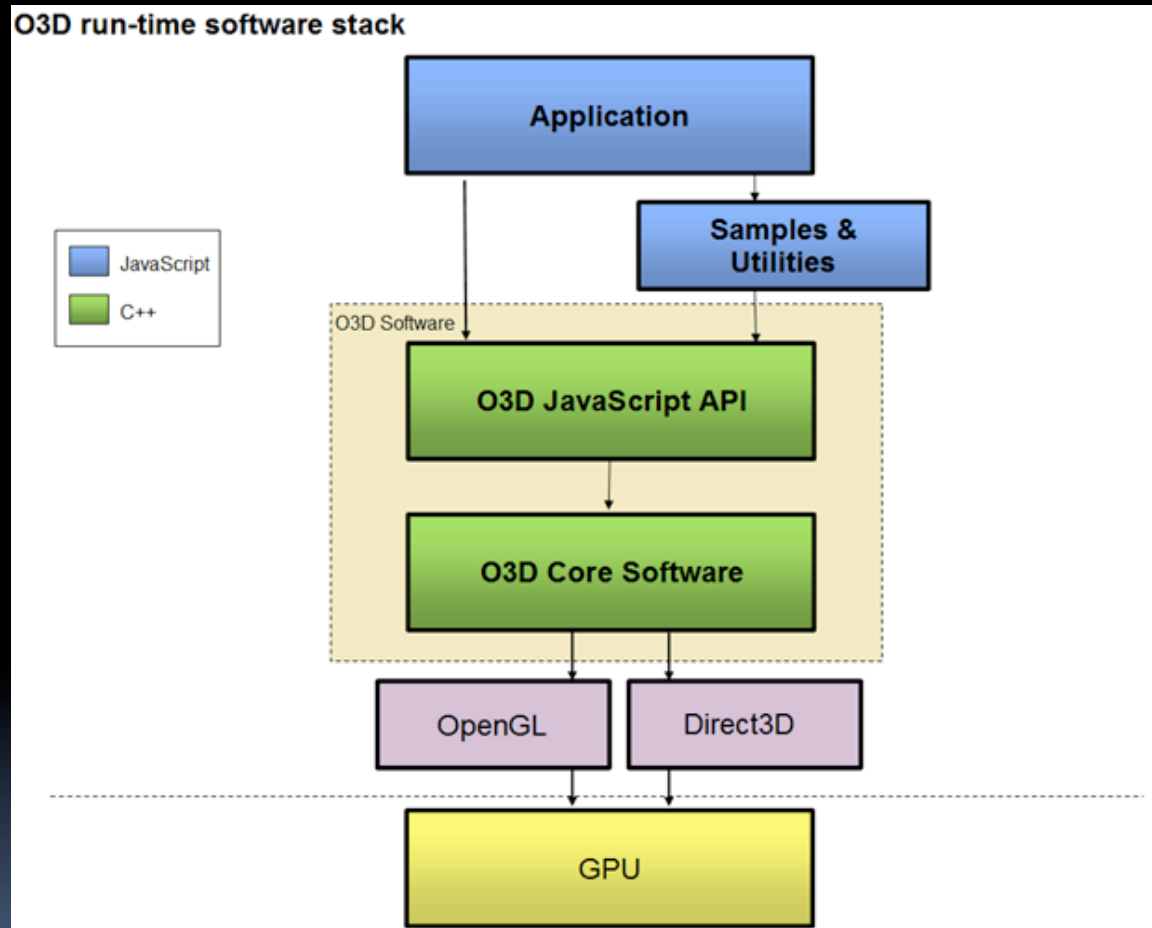
It is not clear which range of 3D formats will be supported

The Google Approach

- Same reasoning and motivation as Mozilla
- Differs in implementation details
- Much further down the line in the development process
- An API is already available with good documentation: O3D
- The approach
 - To develop an OpenGL wrapper within JavaScript
 - Effectively, O3D is a JavaScript extension
 - Plug-ins are required

Google has opted to support the COLLADA format for 3D contents, but this still needs conversion to O3D notation

The O3D Solution



Source: <http://code.google.com/apis/o3d/docs/techoverview.html>

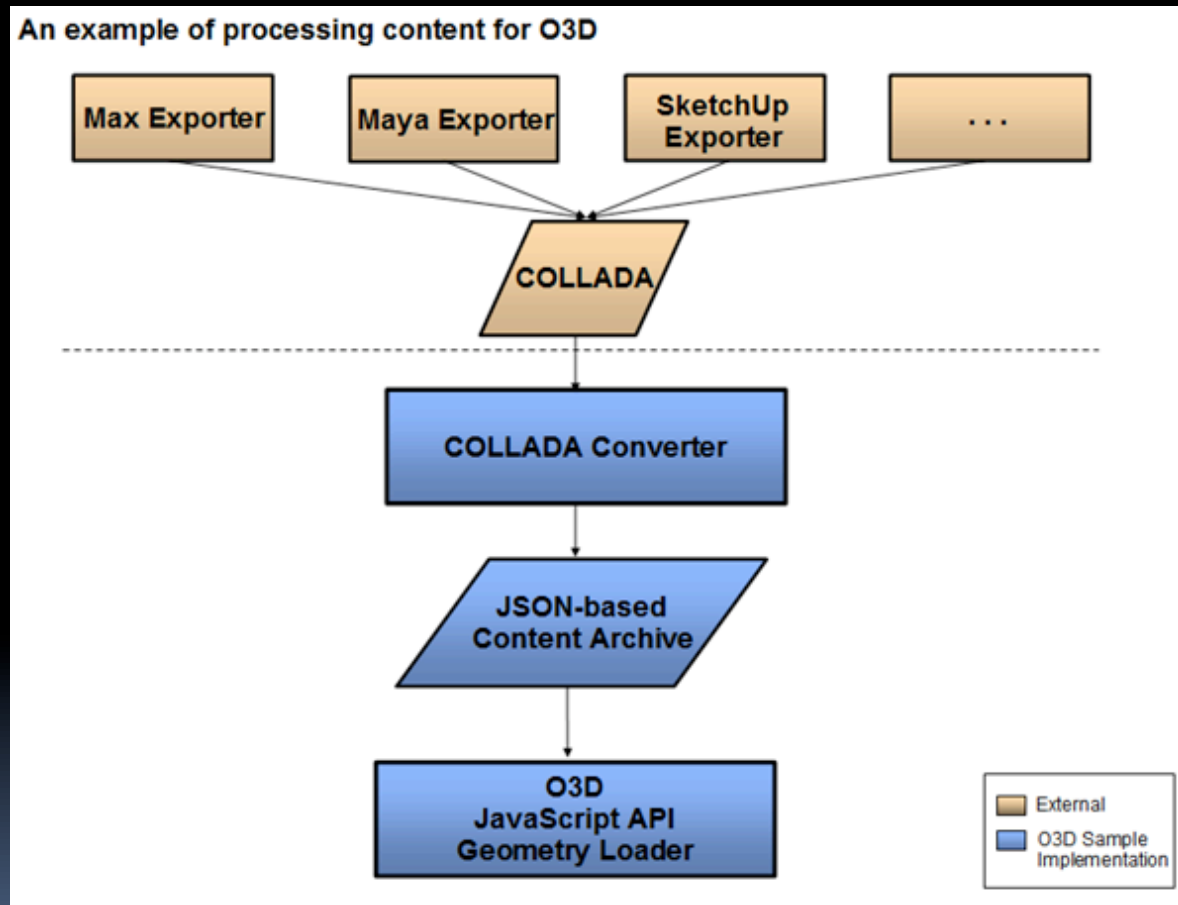
The O3D JavaScript Extension

- O3D is an open source JavaScript extension
- The O3D API provides interactive web based applications
- Interfaces with OpenGL
- Harnesses the power of the client machine
- Available for Windows, Mac, Linux

Creating a 3D Scene

- With O₃D we can:
 - Import 3D scene models; or
 - Create *transform graphs* from scratch
- Transform graphs specify the colour, normals, effects, and the position of objects in 3D space
- We can also use a combined approach, importing some models and adding others from scratch

Importing 3D Contents to O3D



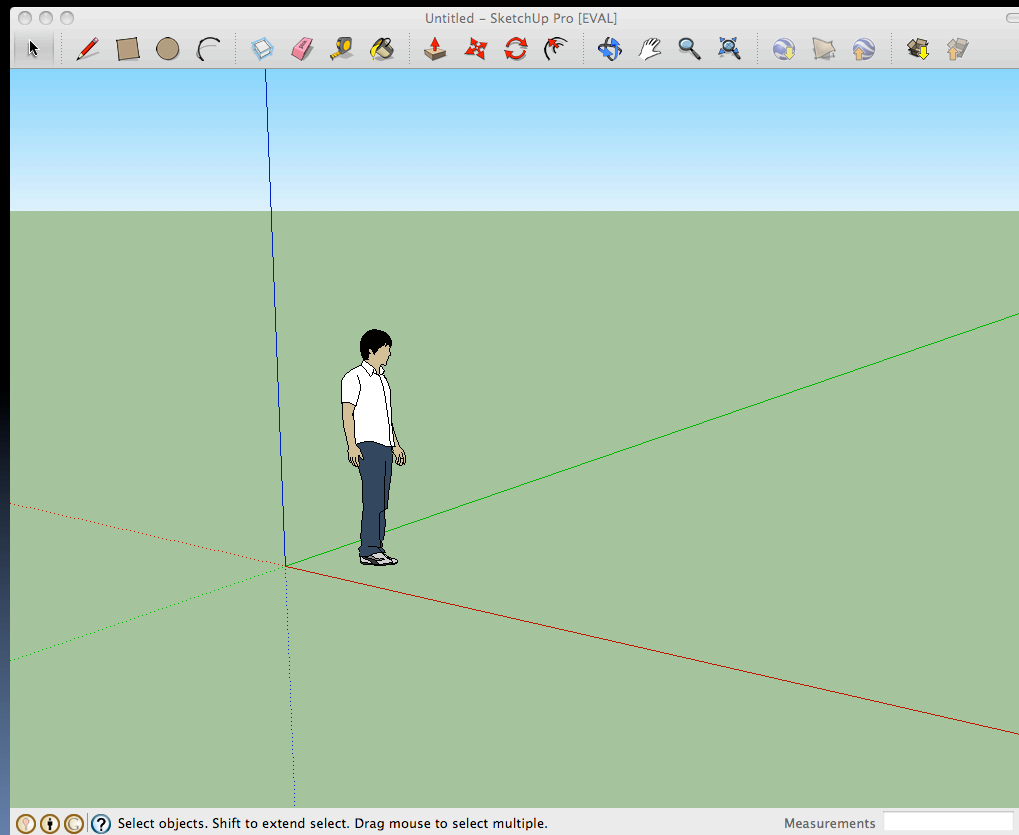
Source: <http://code.google.com/apis/o3d/docs/techoverview.html>

Tools Required

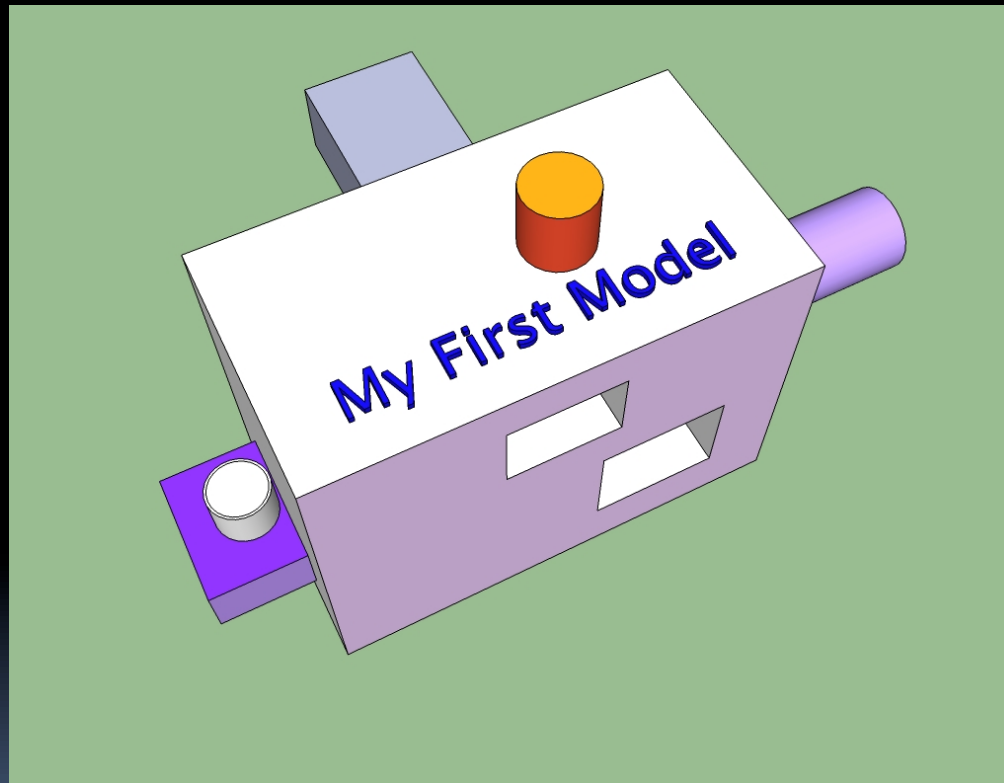
- The O3D plug-in
- A 3D modelling package that can export to COLLADA format
- A COLLADA file converter to *.o3dtgz* (provided by O3D API)
- An html editor to include O3D JavaScript commands to load and display data

Example: Creating 3D Contents

- Start your favourite 3D modelling package

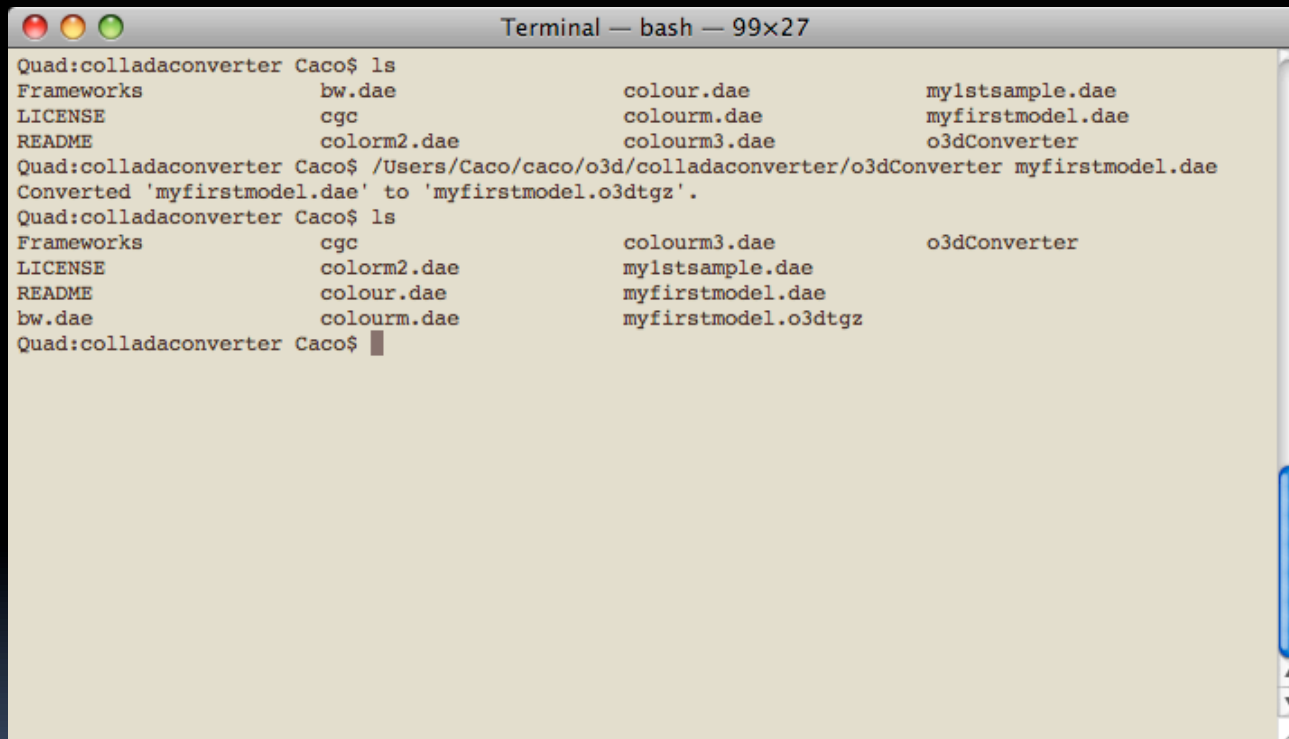


Create a Model and Export to COLLADA Format



The exported file has the extension *.dae* (Digital Asset Exchange)

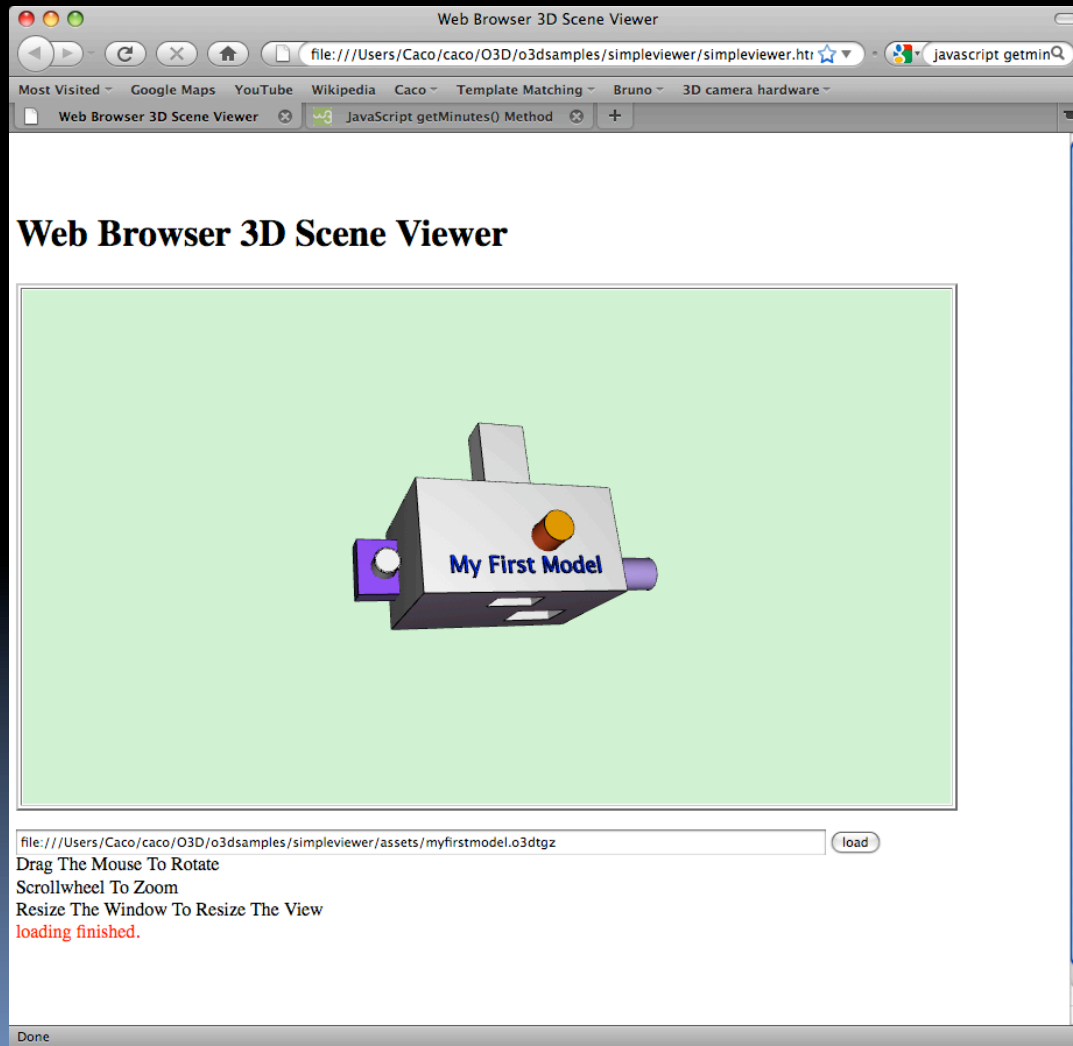
Convert the Model to JSON o3dtgz



```
Terminal — bash — 99x27
Quad:colladaconverter Caco$ ls
Frameworks          bw.dae              colour.dae          my1stsample.dae
LICENSE             cgc                 colourm.dae         myfirstmodel.dae
README              colorm2.dae         colourm3.dae        o3dConverter
Quad:colladaconverter Caco$ /Users/Caco/caco/o3d/colladaconverter/o3dConverter myfirstmodel.dae
Converted 'myfirstmodel.dae' to 'myfirstmodel.o3dtgz'.
Quad:colladaconverter Caco$ ls
Frameworks          cgc                 colourm3.dae        o3dConverter
LICENSE             colorm2.dae         my1stsample.dae
README              colour.dae          myfirstmodel.dae
bw.dae              colourm.dae         myfirstmodel.o3dtgz
Quad:colladaconverter Caco$ █
```

The file has now the extension *.o3dtgz* (Tar-Gnu-Zipped) and can then be displayed on the browser

Viewing and Interacting with the Model on a Web Browser



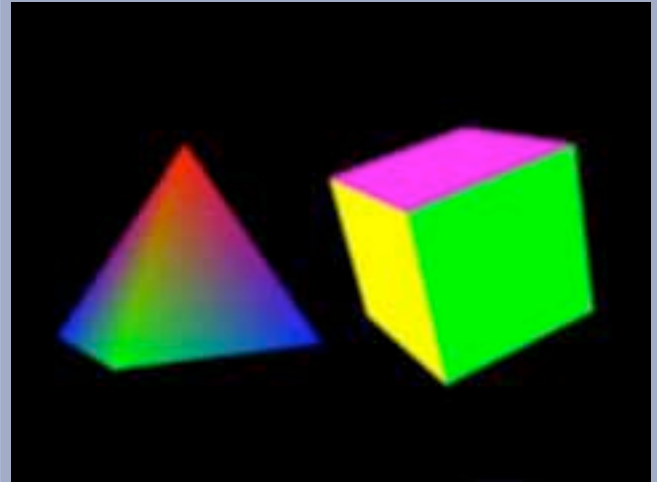
Not a Single OpenGL Low Level Construct is Required

```
glBegin(GL_TRIANGLES);

glColor3f(1.0f,0.0f,0.0f); // Red
glVertex3f( 0.0f, 1.0f, 0.0f); // Top Of Triangle
glColor3f(0.0f,1.0f,0.0f); // Green
glVertex3f(-1.0f,-1.0f, 1.0f); // Left Of Triangle
glColor3f(0.0f,0.0f,1.0f); // Blue
glVertex3f( 1.0f,-1.0f, 1.0f); //Right Of Triangle

.....

glEnd();
```



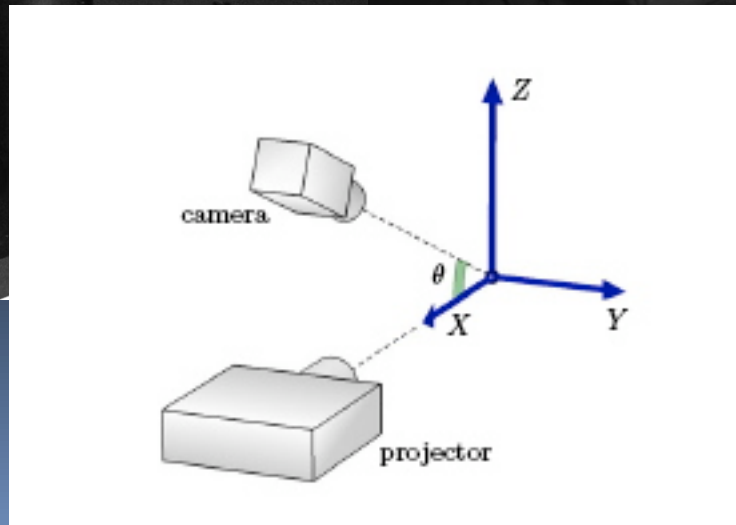
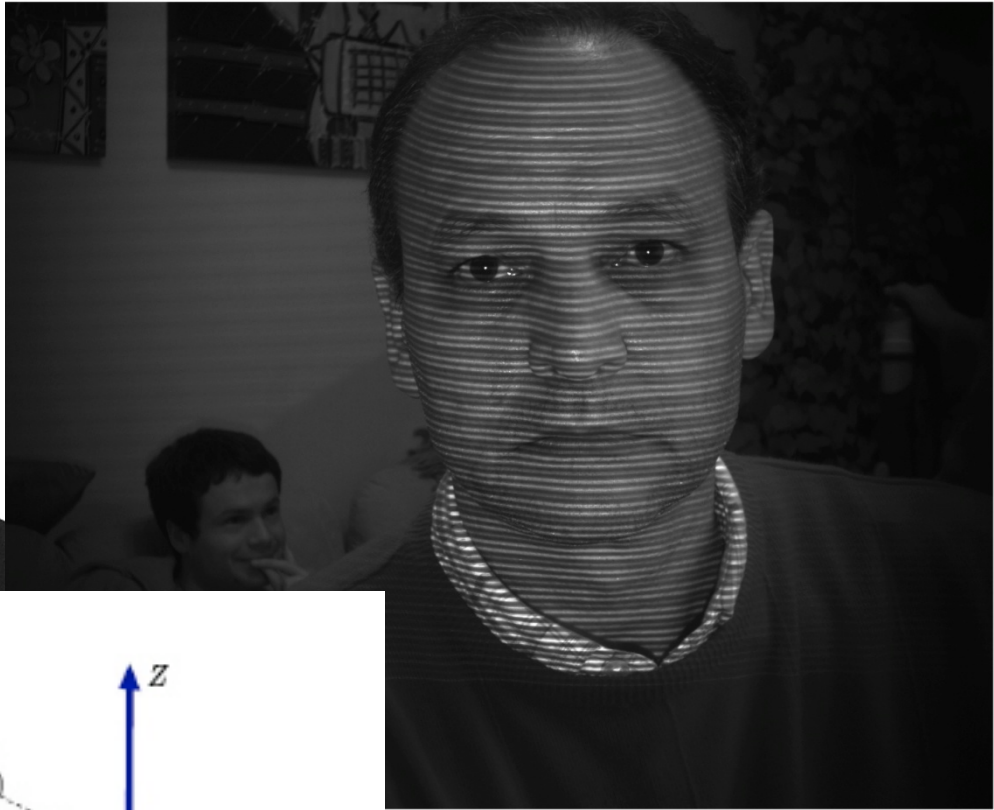
However, O3D provides the equivalent OpenGL constructs (*transform graphs*) as above giving the developer the option for a combined approach

03D Sample Code

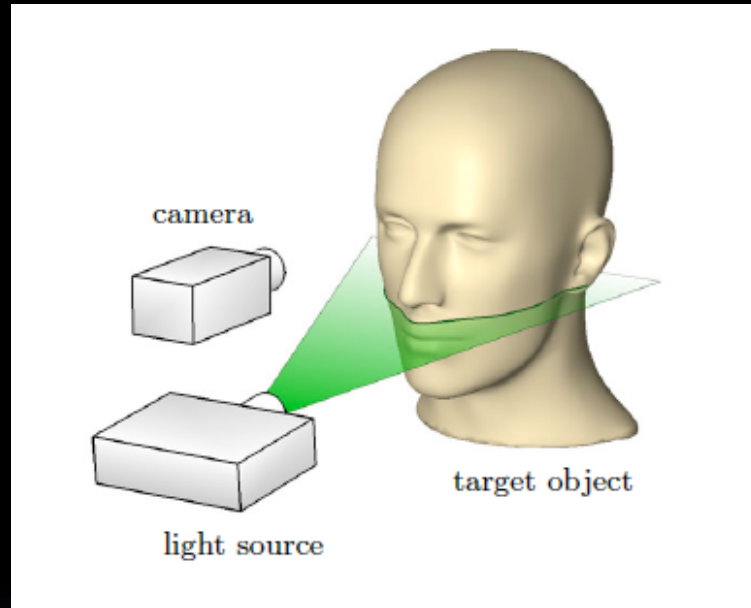
```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
  <title>
    Web Browser 3D Scene Viewer
  </title>
</head>
<body onload="init();" onunload="uninit();">
<script type="text/javascript" src="./o3djs/base.js"></script>
<script type="text/javascript">
... ..
function onRender() {
  g_viewInfo.clearBuffer.clearColor = [ 0.8, 0.95, 0.8, 1 ];
  setClientSize();
}
... ..

function initLoadFile(clientElements) {
  var path = window.location.href;
  var index = path.lastIndexOf('/');
  path = path.substring(0, index+1) + 'assets/myfirstmodel.o3dtgz';
  var url = document.getElementById("url").value = path;
  g_loadingElement = document.getElementById('loading');
  ... ..
}
... ..
</script>
... ..
</body>
</html>
```

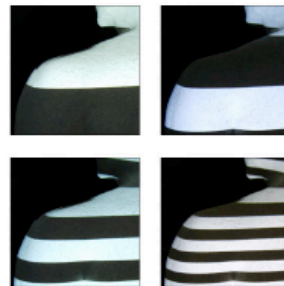
A 3rd Way of Acquiring 3D Models



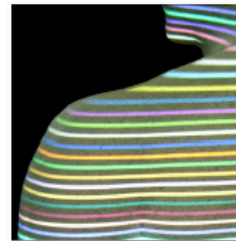
A Number of Alternative Projection Methods Can Be Used



The camera and projector can be parallel or angled, and coding schemes can vary



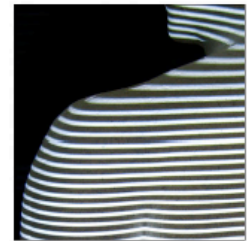
(a) time-multiplexing



(b) colour coding

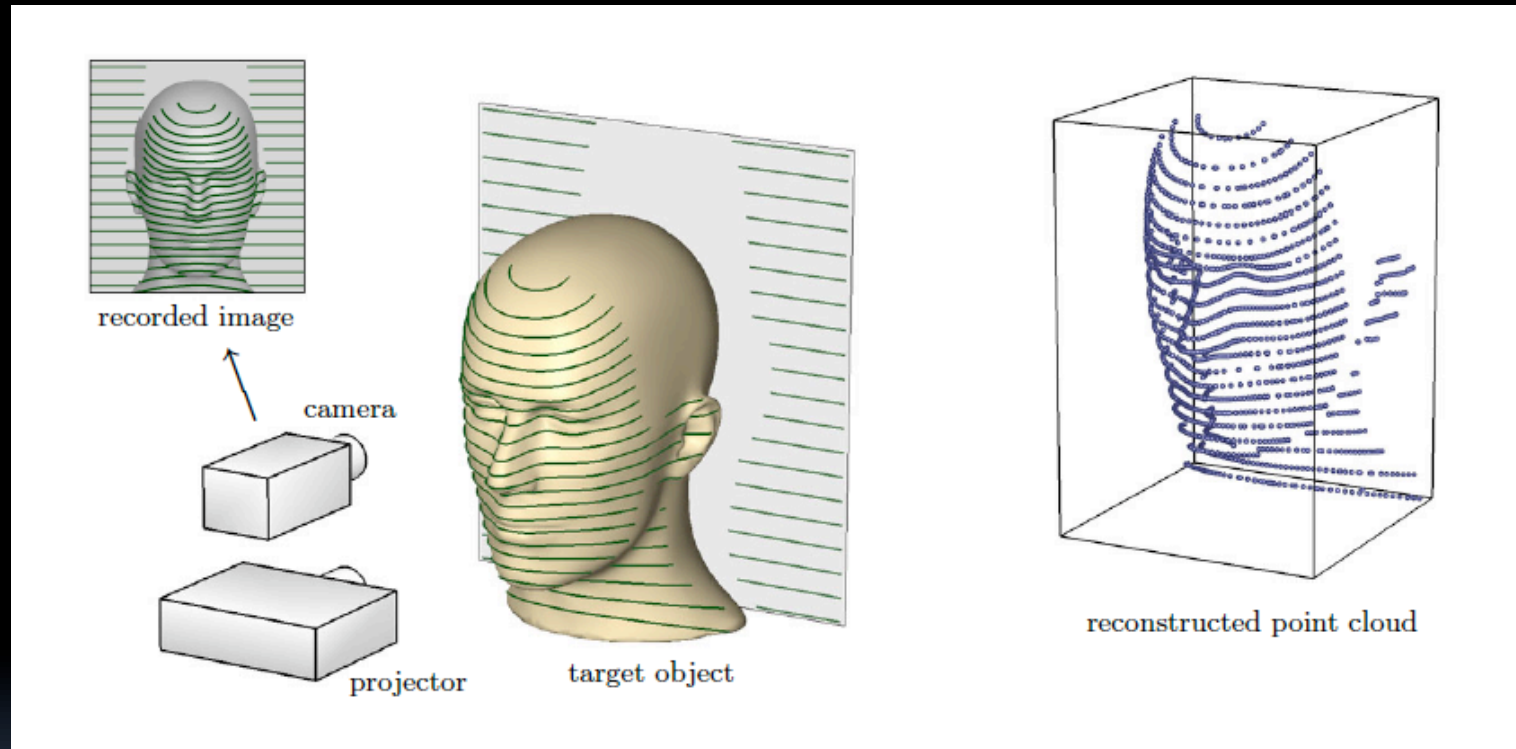


(c) variable width



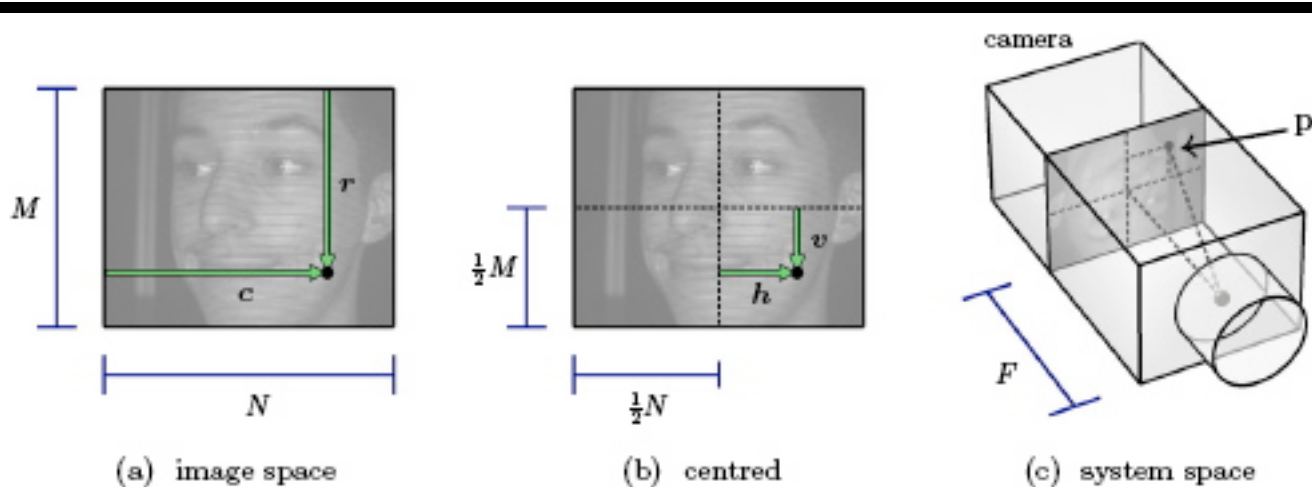
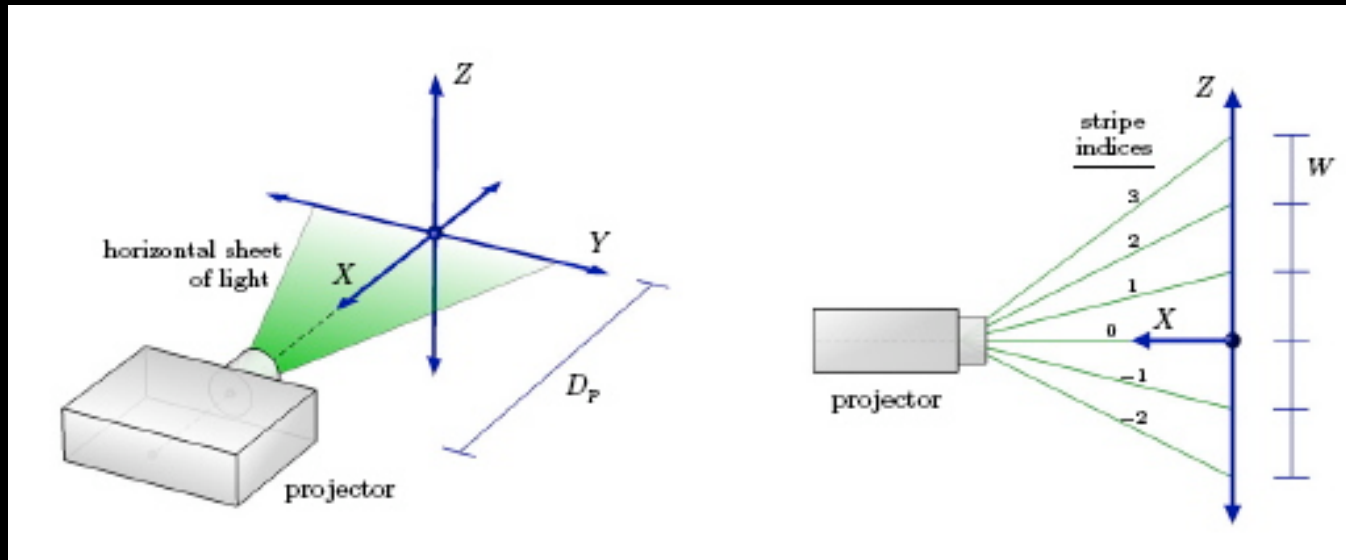
(d) uncoded stripes

Our 3D Striper Method

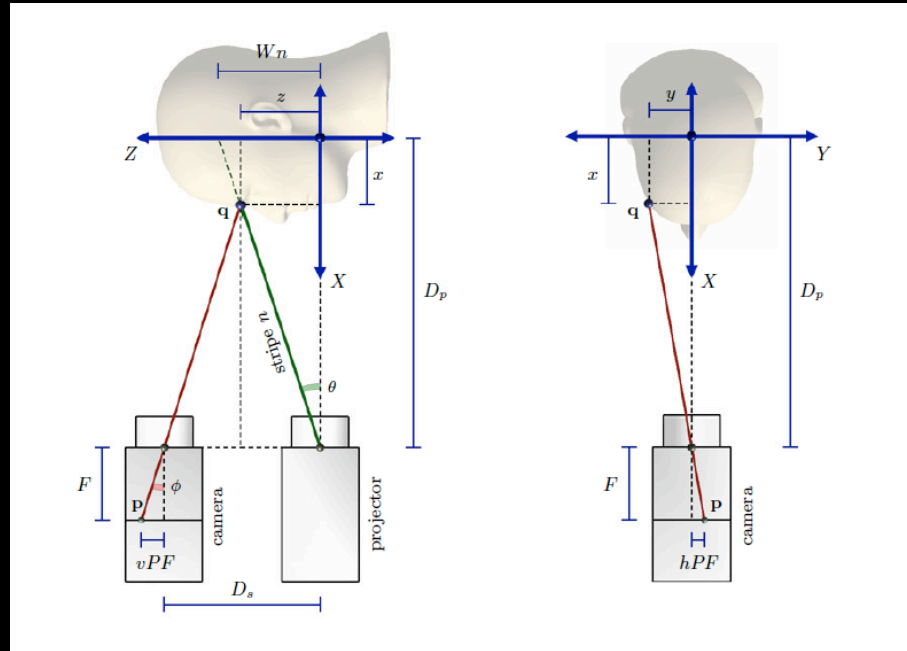


A series of parallel stripes are projected onto a surface of an object. Stripe information is used to reconstruct a point cloud in 3D space

The projector determines the centre of the system



Geometrical Relationships

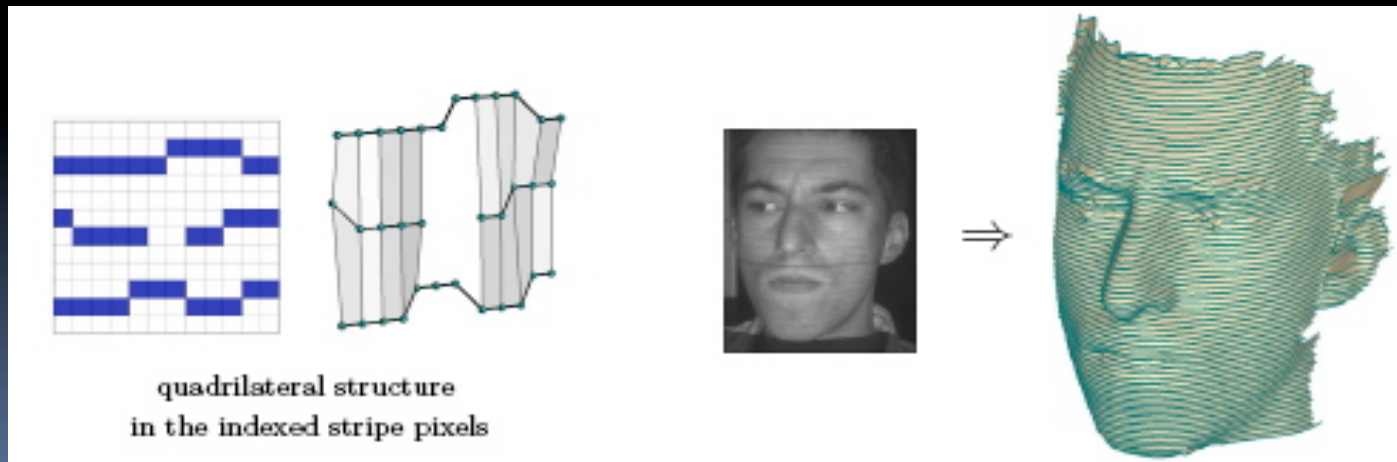
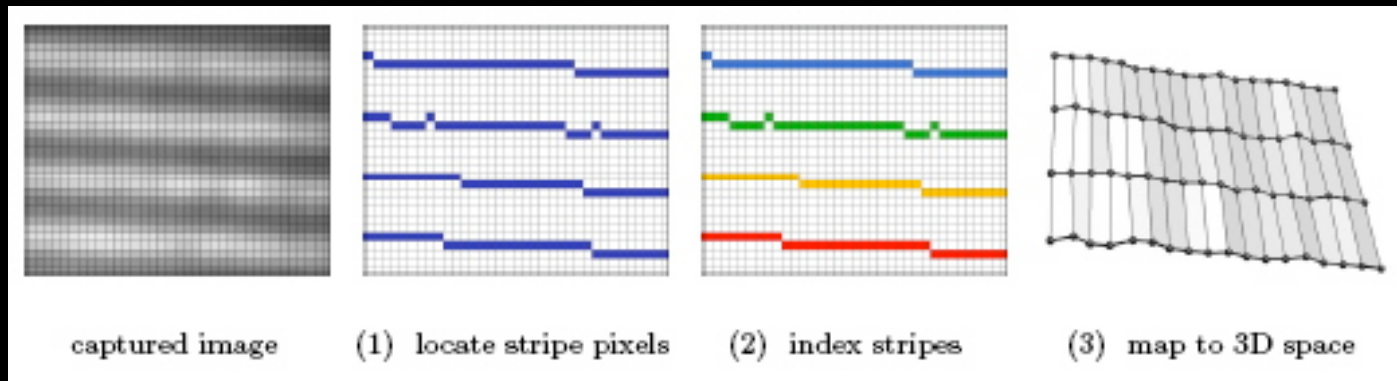


$$z = \frac{W_n}{D_p} \left(D_p - D_p + \frac{D_p D_s}{vP D_p + W_n} \right) = \frac{W_n D_s}{vP D_p + W_n}$$

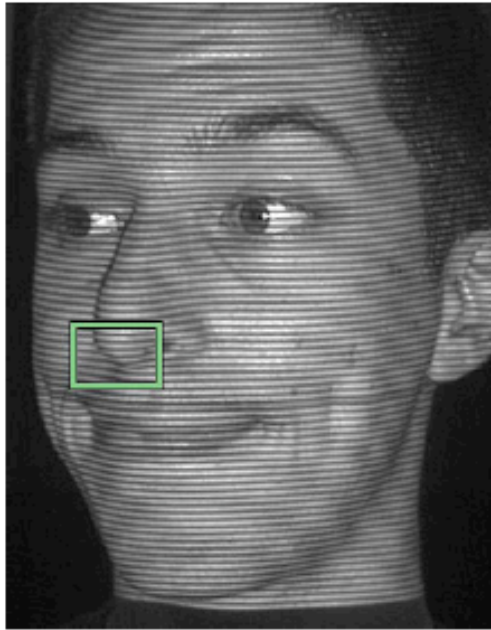
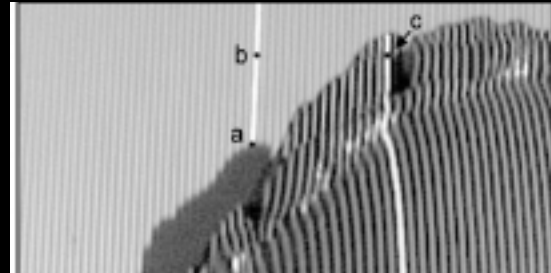
$$\frac{W_n}{D_p} (D_p - x) = D_s - vP (D_p - x) \implies x = D_p - \frac{D_p D_s}{vP D_p + W_n}$$

$$\frac{y}{D_p - x} = \frac{hPF}{F} \implies y = hP (D_p - x) = \frac{hP D_p D_s}{vP D_p + W_n}$$

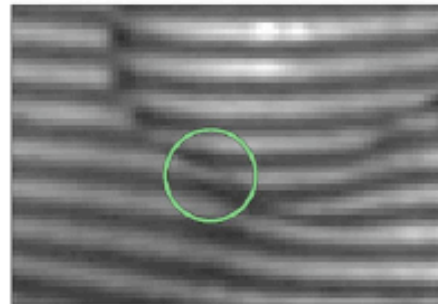
Stripe Indexing and Generation of 3D Data



Problems with Occlusion



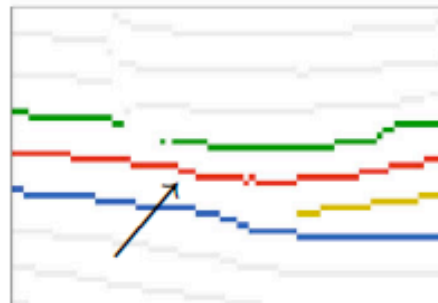
(a) captured image



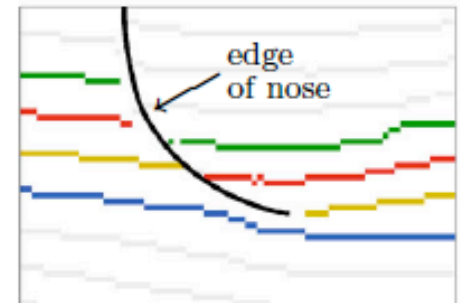
(b) close-up



(c) stripe pixels

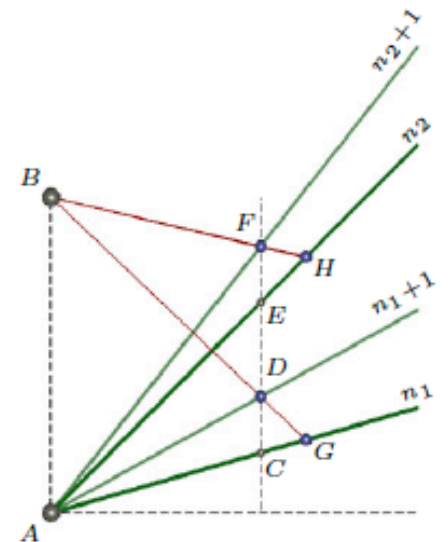
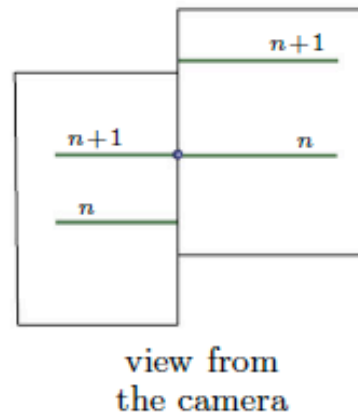
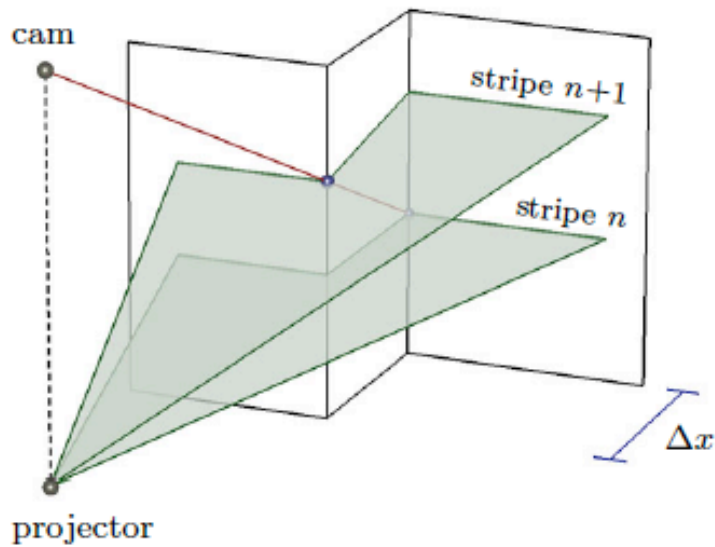


(d) incorrect indexing

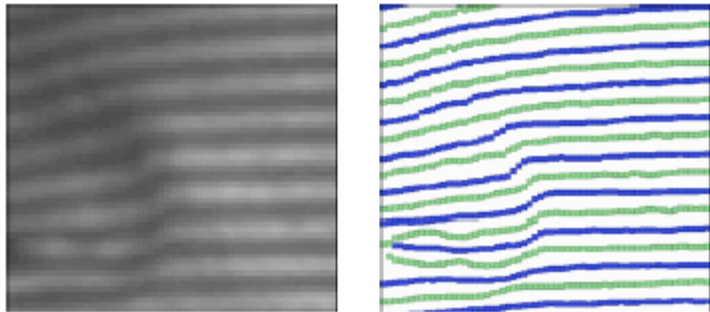
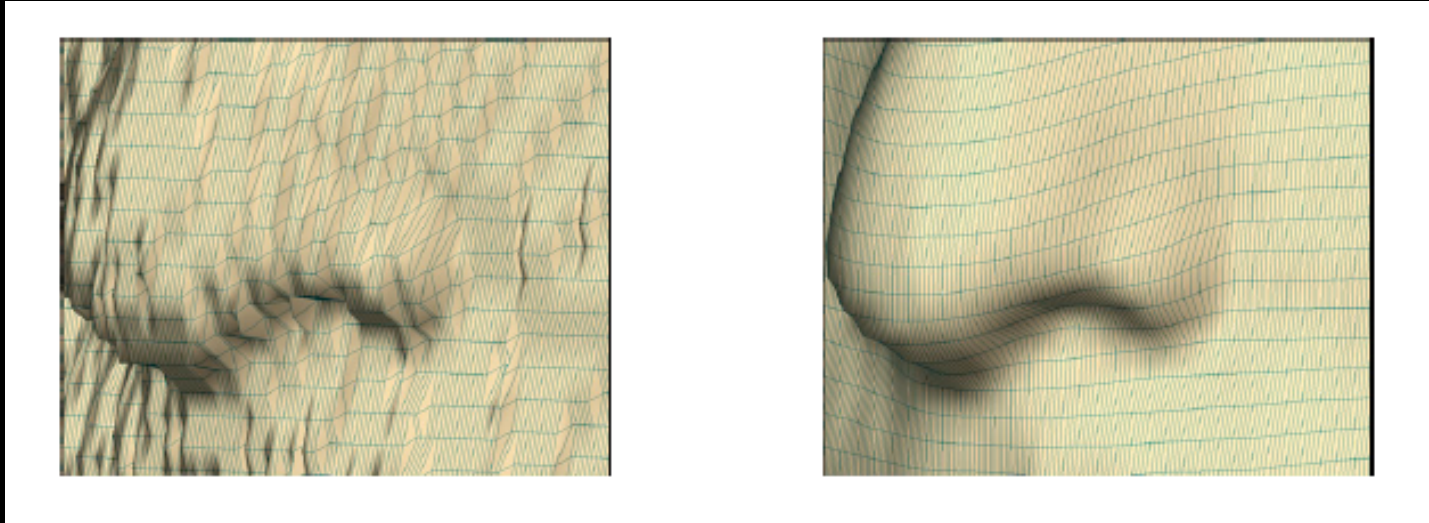


(e) correct indexing

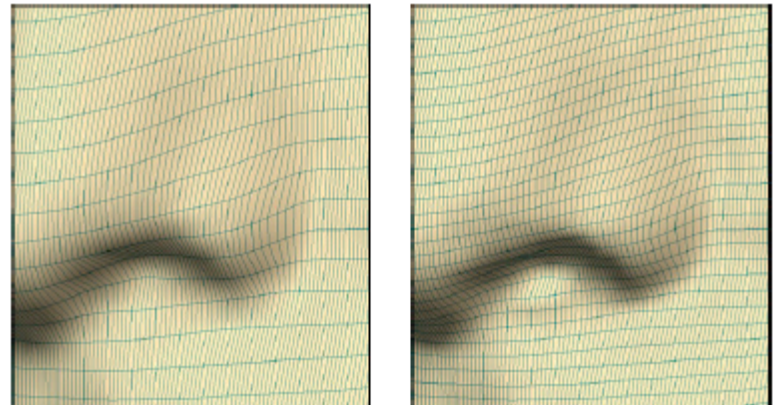
Unsolvable Occlusion Problem



Sub-pixel Estimation

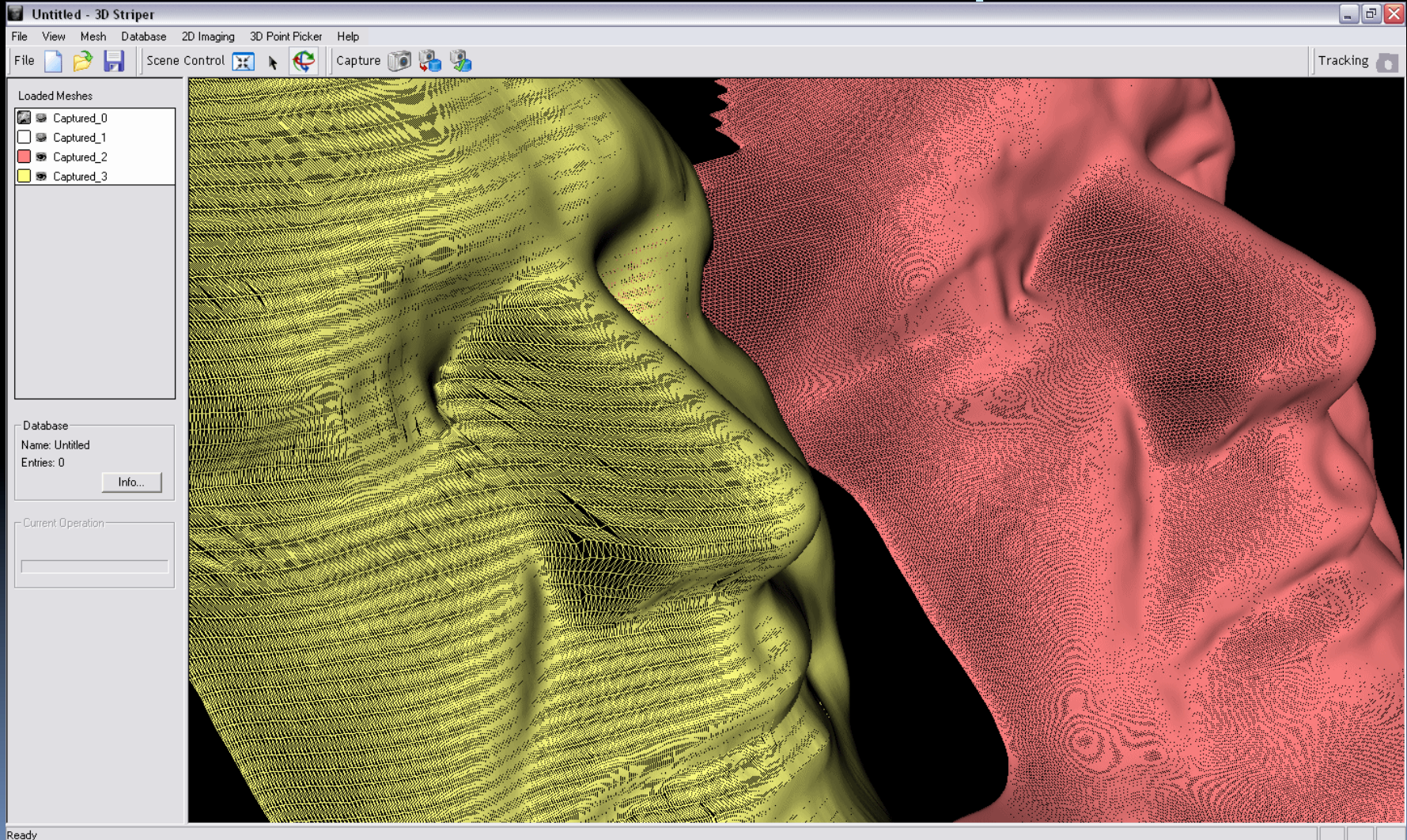


(a) black stripes located between white ones

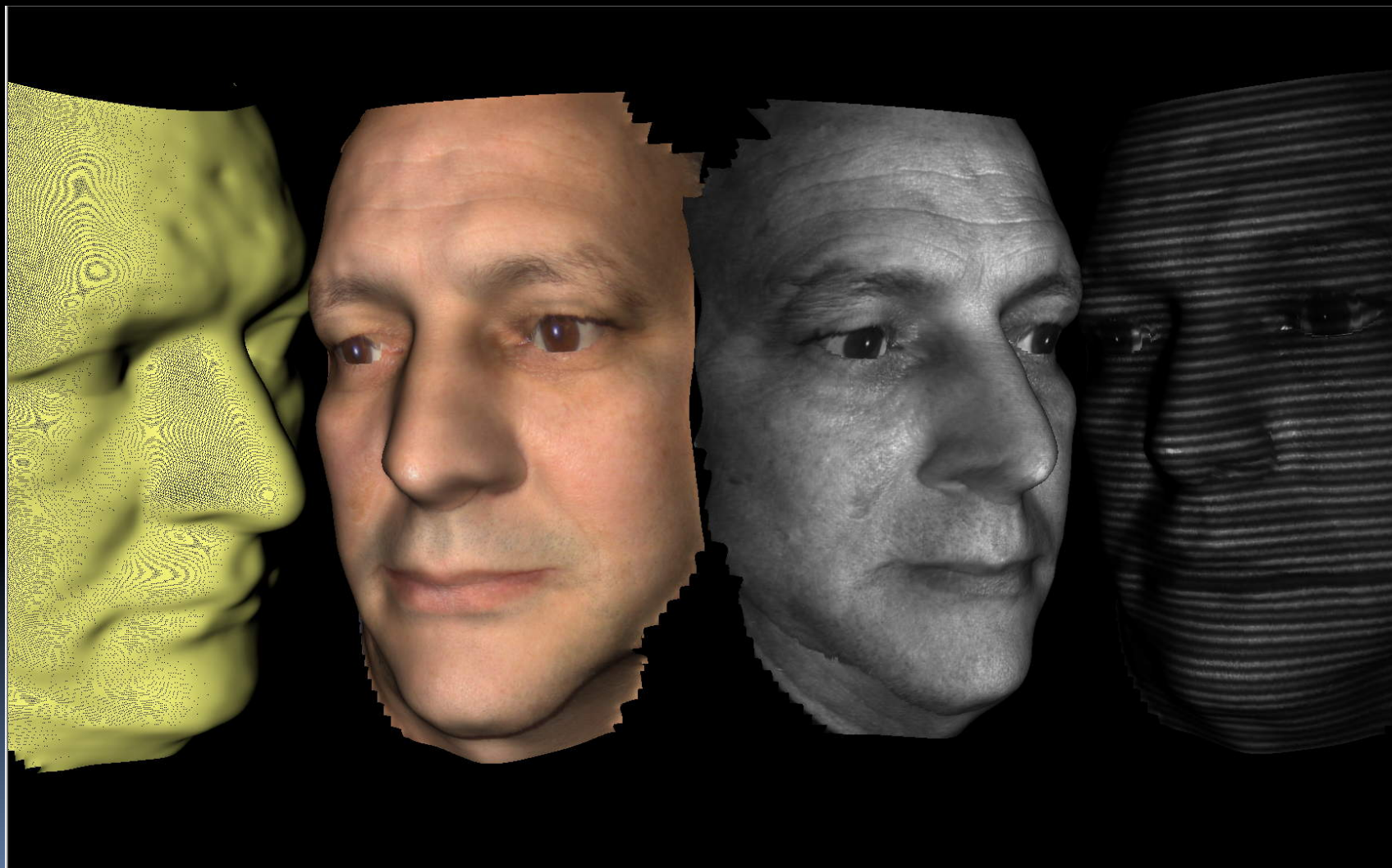


(b) result on the reconstructed surface

The Effects of Computing White and Black Stripes



Texture Mapping




Performance



- 3D reconstruction in 40ms
- Only a single 2D image is required per model
- Can operate at video frame rates
- Also suitable for 3D animation from video frames
- It requires post-processing (hole filling, smoothing, noise removal)
- Sound can be integrated from video footage



Future work

- Hardware issues
 - Embedding the design into DSPs
 - Methods for 3D image compression
 - Built-in exporters
 - 3D data -> COLLADA -> o3dtgz
 - 3D data -> COLLADA
- 

Related work: Feature extraction & recognition

