

Desk-top VR – A Hands-on Approach for Hands-free Experience

Professor Zhijie Xu

Centre for Visual and Immersive Computing

University of Huddersfield, UK

Things about Huddersfield ...

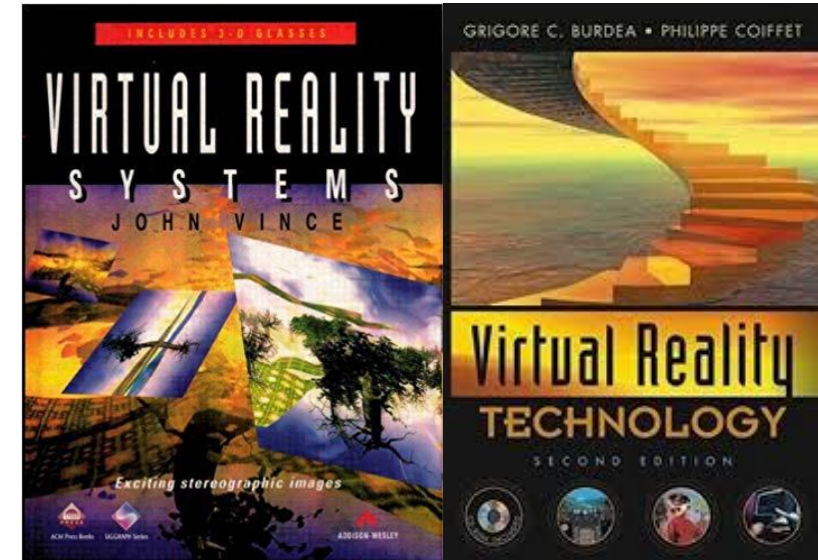


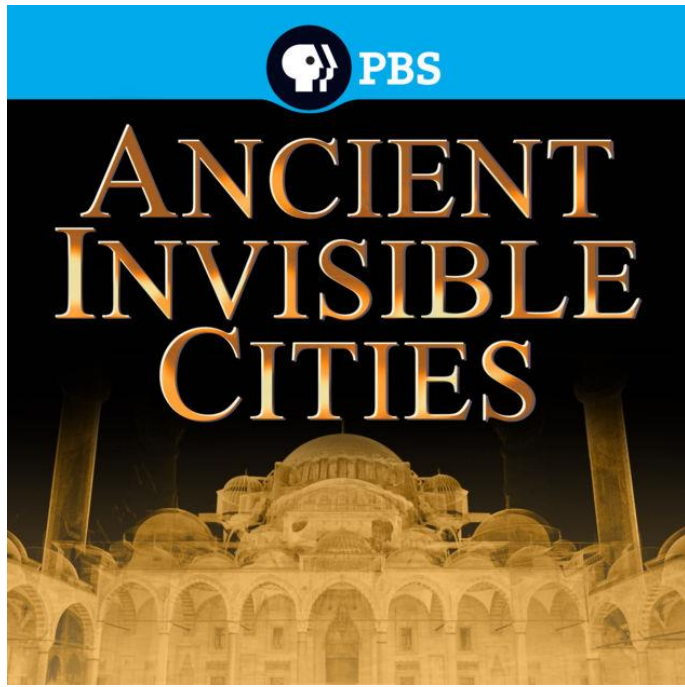
Outline

1. What is VR?
2. What is Desk-top VR?
3. How to develop a Desk-top VR system?
4. What are the potential usages of Desk-top VR?
5. Discussions

1. What is Virtual Reality?

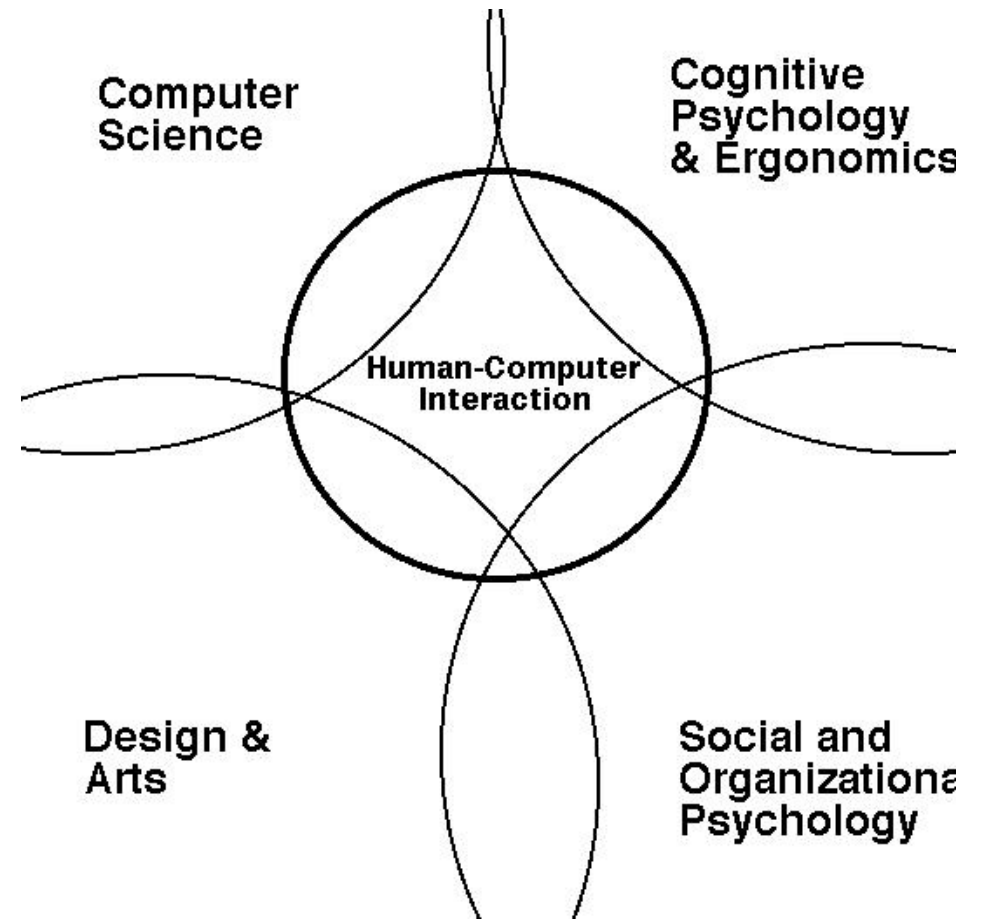
- Is there an official definition for VR?
 - “... a high-end user-computer interface that involves real-time simulation and interaction through multiple sensorial channels, i.e. visual, auditory, haptic, olfactory.”
 - “... VR should be computer-generated, believable, interactive, explorable, immersive.”
- It’s more of an open discussion in science.



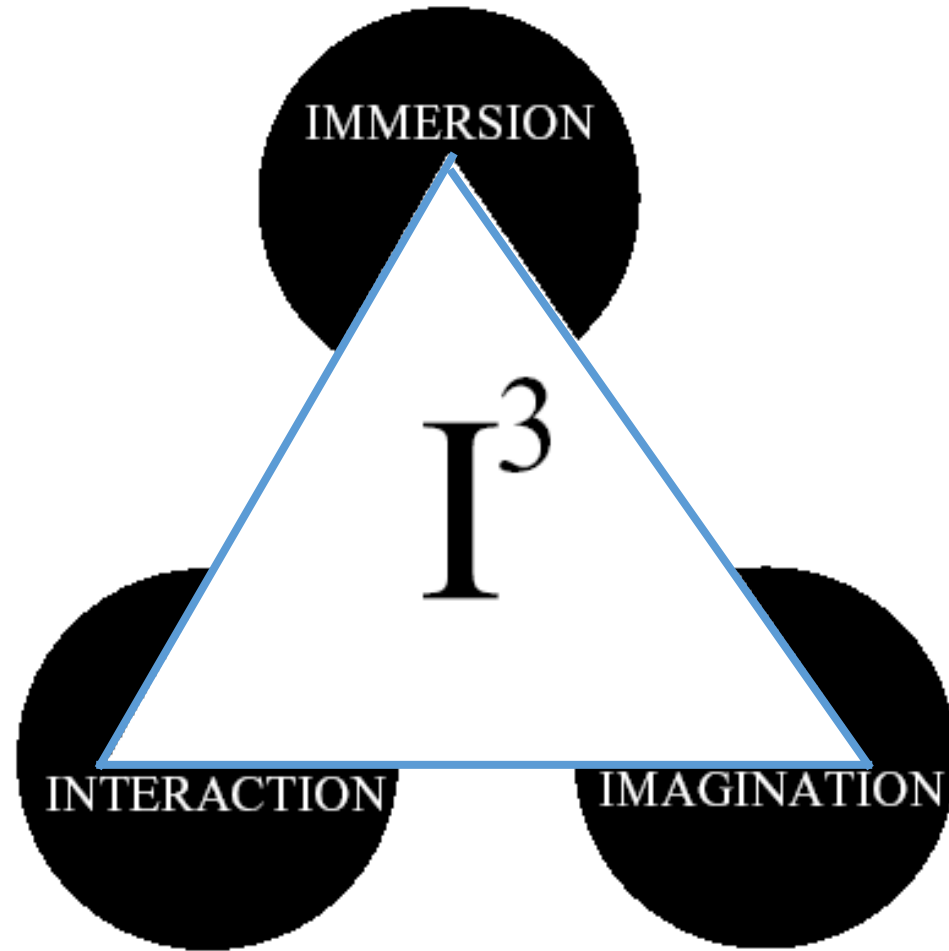


More Cautiously ...

- VR is a **HCI** effort; an integrative system; a new Operating System; an UI Paradigm; a technology idealism.
- My Research Interests:
 - Virtual and Augmented Reality
 - Real-time Graphics
 - Computer Vision and Machine Learning



The Famous VR Triangle – does it have to be equilateral?



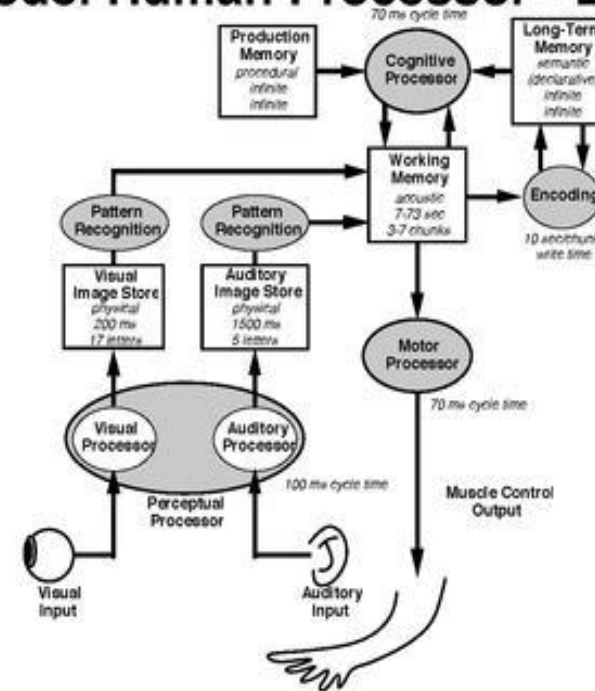
<http://swab.cityu.edu.hk/sm2203/oldclass2007/lab1/toc02/index.html>

HCI: An Empirical Research Perspective

- By I. Scott MacKenzie

Scale (sec)	Time Units	World (theory)
	Months	
	Weeks	
	Days	SOCIAL BAND
	Hours	
	10 min	
	Minutes	RATIONAL BAND
	10 sec	
	1 sec	
	100 ms	COGNITIVE BAND
	10 ms	
	1 ms	
	100 μ s	BIOLOGICAL BAND

The Model Human Processor - Diagram



Modified from Card, Moran, & Newell (1983)

As VR Practitioners, we are more or less here...

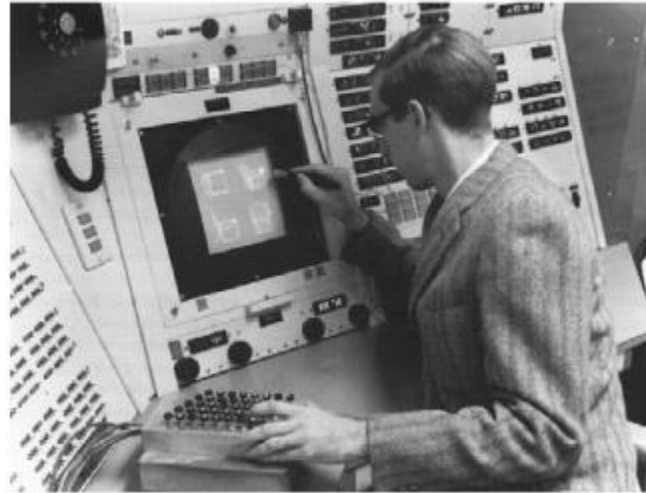
Questions Posed to HCI/VR

- How do you design interfaces to systems for:
 - Users with special needs, i.e. disabled, children, elderly...
 - Culture and international diversified
 - Cognitively and physically varied
- But, to a degree, keeping universal usability
- ...

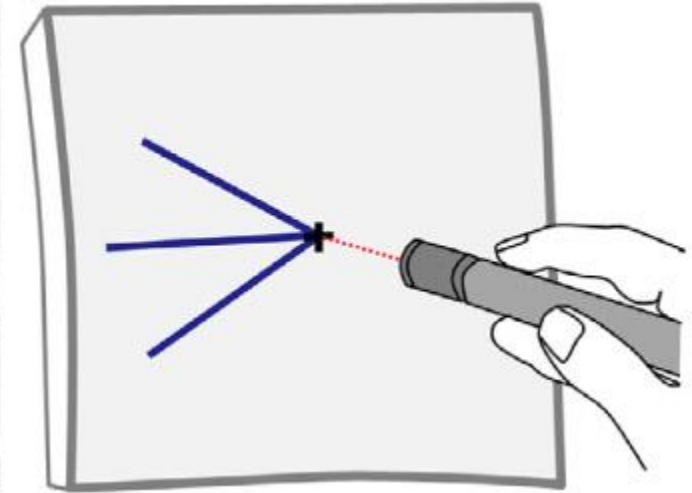
A Pioneer: Ivan Sutherland's Sketchpad (1962)



(a)



(b)



- “With Sketchpad, commands were not typed. Users did not “write letters to” the computer. Instead, objects were drawn, resized, grabbed and moved, extended, deleted—directly, using the light pen. Object manipulations worked with constraints to maintain the geometric relationships and properties of objects ...” – Ivan’s report

Early Party Time (first wave - Virtual Reality in the late 80s)

Efforts for making profits:

- The first commercial VR systems appeared in the late 80s produced by VPL Co. (California):
- The VPL “Data Glove” and
- The VPL “Eye Phone” HMD



Early Party Time (first wave - Virtual Reality in the early 90s)

✓ PC boards still very slow
(7,000 – 35,000 polygons/sec);

✓ First turnkey VR system –
Provision 100 (Division Ltd.)

✓ Emergence of faster graphics
rendering architectures at UNC
Chapel Hill:

“Pixel Planes”;

Later “Pixel Flow”;

Emergence of first commercial
Toolkits:

✓ **WorldToolKit (Sense8 Co.);
Used in UoH between 98 and 04**

✓ VCToolkit (Division Ltd., UK);

✓ Virtual Reality Toolkit VRT3
(Dimension Ltd./**Superscape**,
UK);

✓ Cyberspace Developer Kit
(Autodesk)

✓ VRML and Java3D

35,000
polygons/sec;
\$64,000 (including
texture generator,
tracker, 3-D audio,
HMD and
software)



Provision 100 VR turnkey system (Division Ltd., UK)

Extracts from my PhD project (1996)

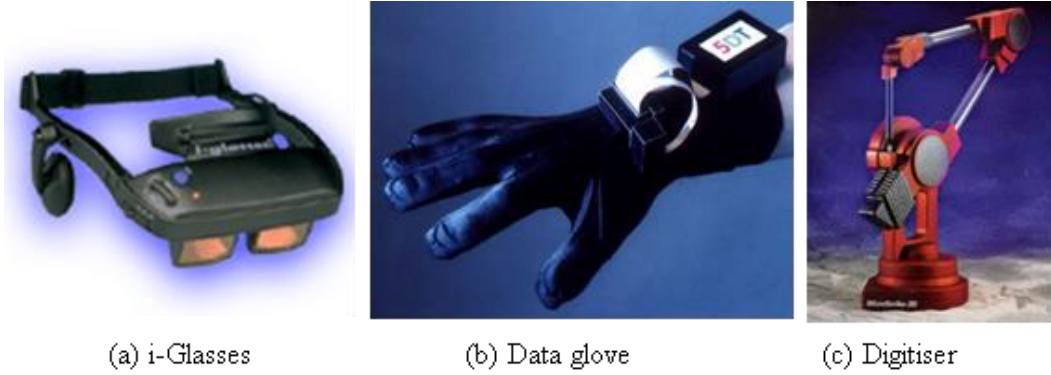


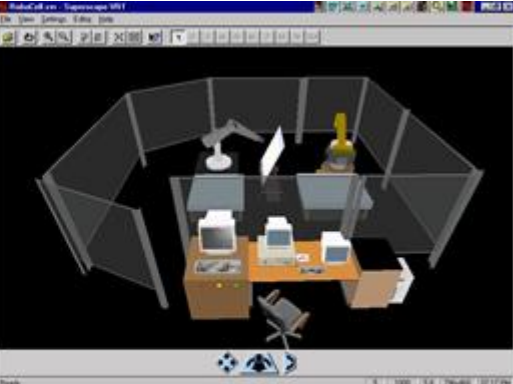
Figure 4.2 VR devices used in the KAMVR system



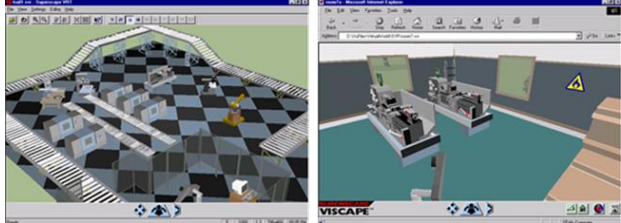
Figure 4.7 KAMVR system workbench



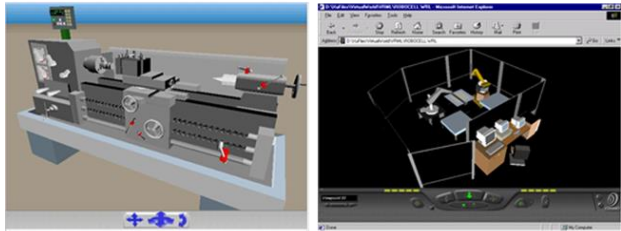
Figure 4.16 (a) Real robot cell (Photo)



(b) Virtual robot cell



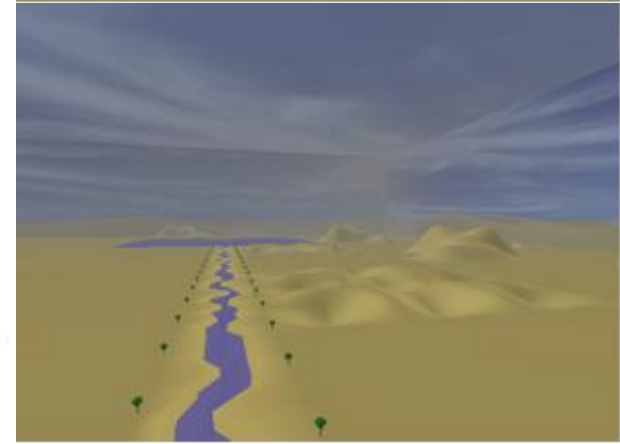
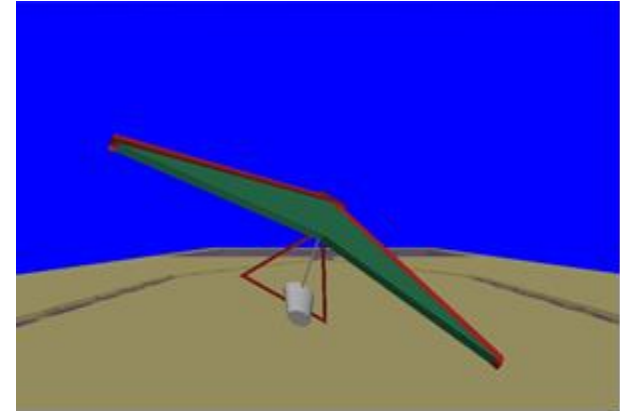
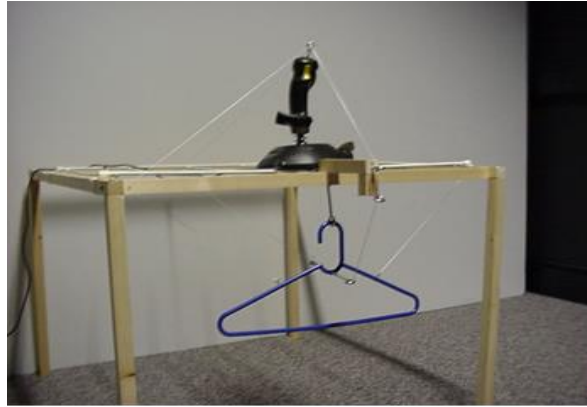
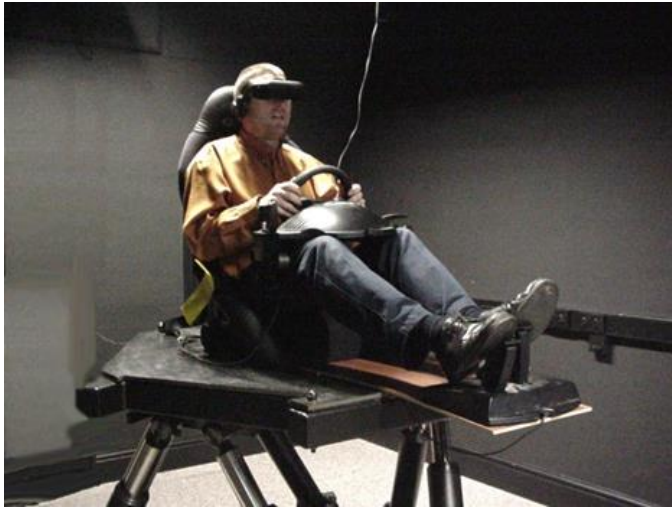
(a) Superscape Visualiser (b) Superscape Viscape



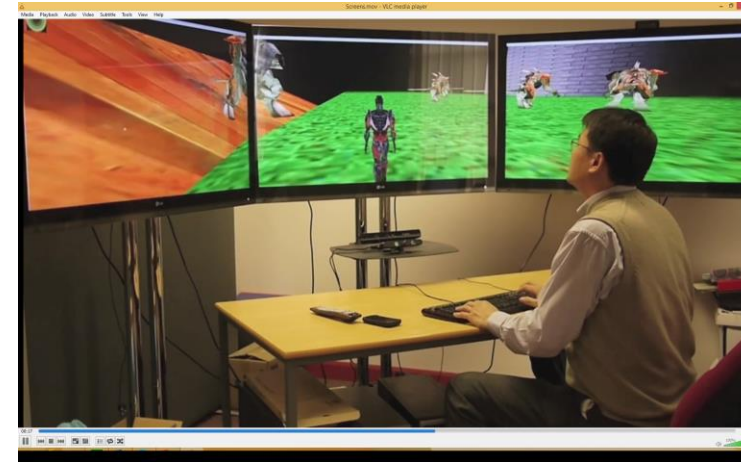
(c) Superscape 3D Control (d) COSMO VRML Player

Figure 4.3 Environment Visualisers

Virtual Glider Simulator @ UoH (2003)

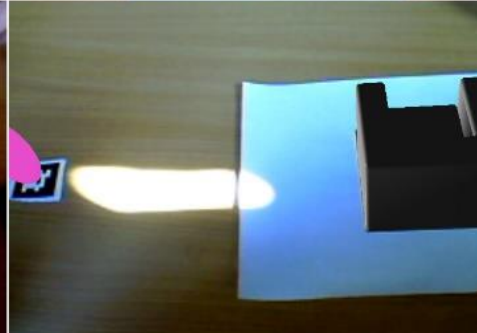
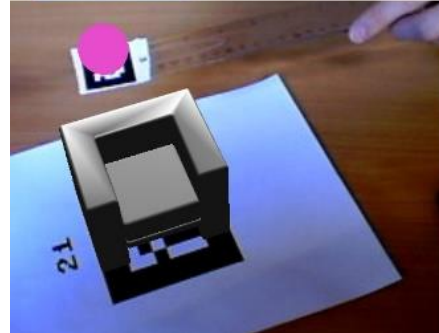
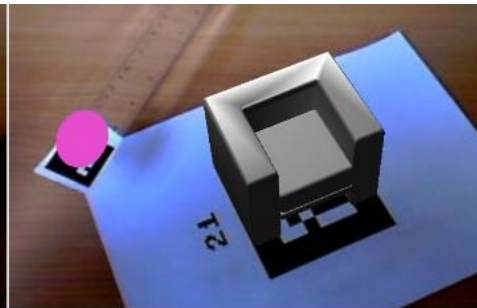
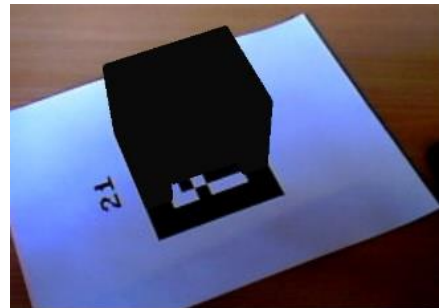


Neuropsychological Rehabilitation @ UoH (2003)



Virtual reality for neuropsychological diagnosis and rehabilitation: A survey
August 2003. DOI:
10.1109/IV.2003.1217973
Conference: Information Visualization, 2003. IV 2003.
<https://ieeexplore.ieee.org/document/1217973>

Hybrid AR Illumination @ UoH (2006)



2. Types of Virtual Reality

- **Fully immersive**
- **Non-immersive**
- Collaborative
- Web-based
- **Augmented reality**

Immersive VR

- Purposely designed input/output equipment
- Physically isolated from the real environment
- Real-time interaction
- Feeling of presence



Non-immersive (Desktop) VR

- Use of a monitor to display the virtual world
- Window on the world (Tardis?)
- Does not require special hardware
- Low cost

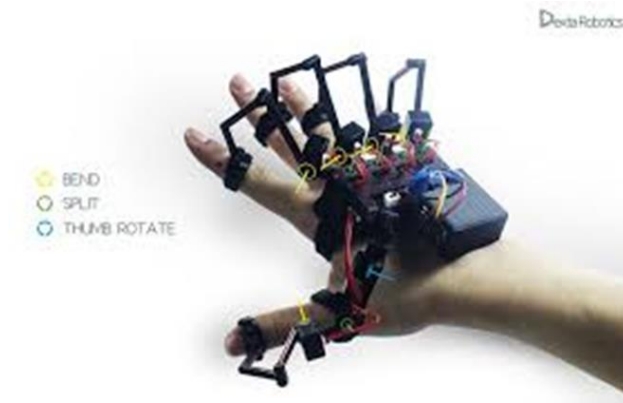


Augmented Reality

- Registered and non-registered hybrid display
- “Hot” in mobile solutions
- Huge potentials



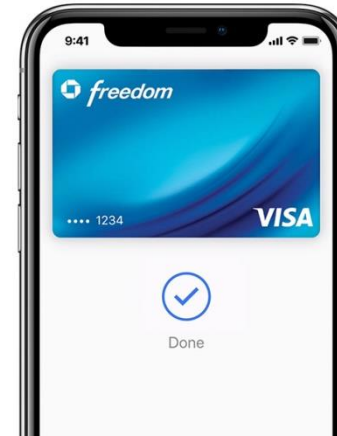
VR Systems (Inputs, Outputs, Software)





Off-the-shelf products

- Kinect SDK
- Eye Tracker – GP3 GazePoint
- Leap Motion
- Emotiv BMI
- VR Headsets (HMDs)
- ...

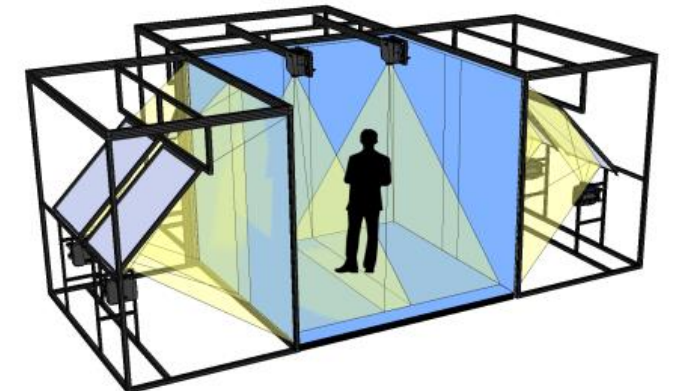
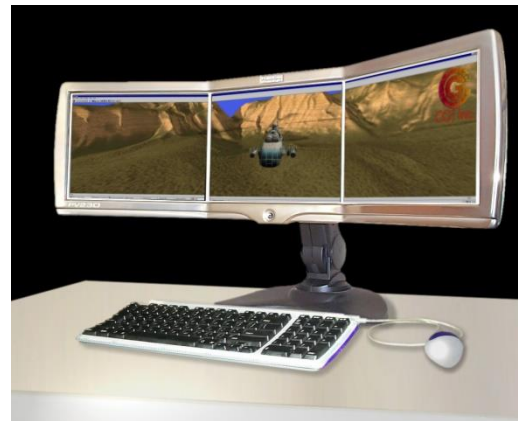
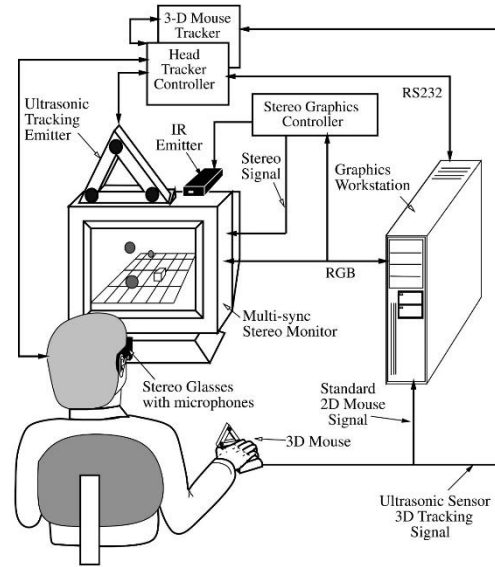


VR Hardware

- Based on sensory channel accessed:
 - Visual interfaces (hi-res, bio-convincing)
 - Tracking interfaces (motion, eye/gaze)
 - Auditory interfaces (3D spatial localization)
 - Haptic interfaces (touch and force)
 - Olfactory interfaces (smell, electric-nose)
- Based on process types:
 - Input devices
 - Output devices

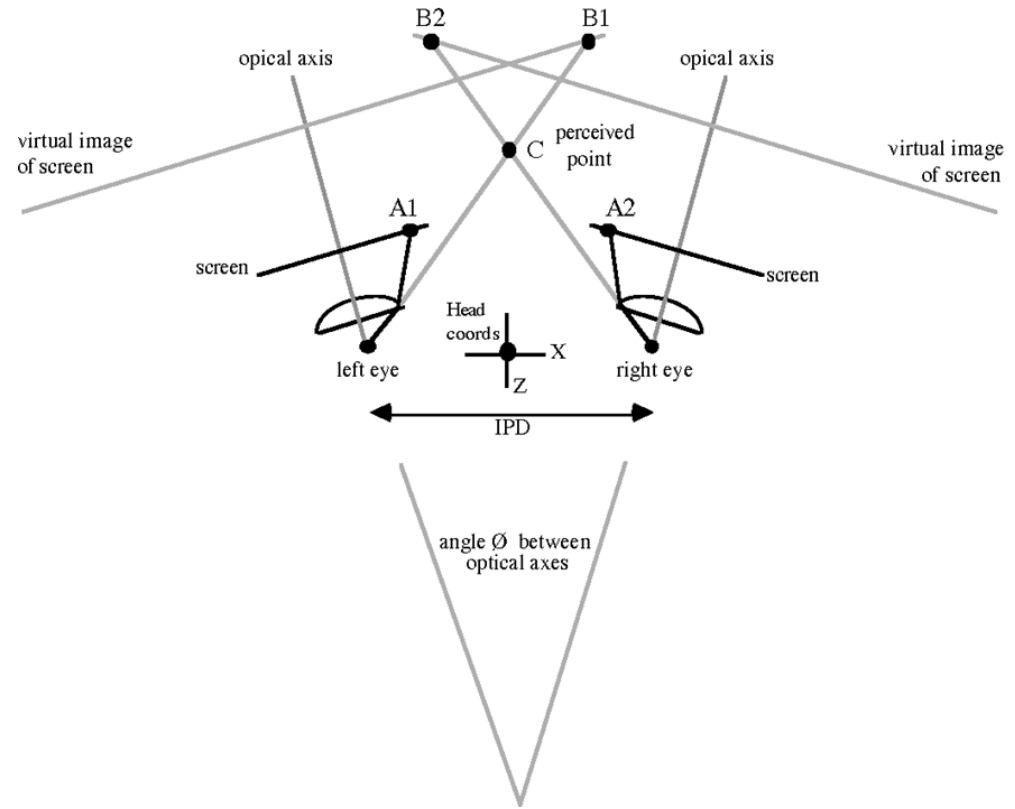
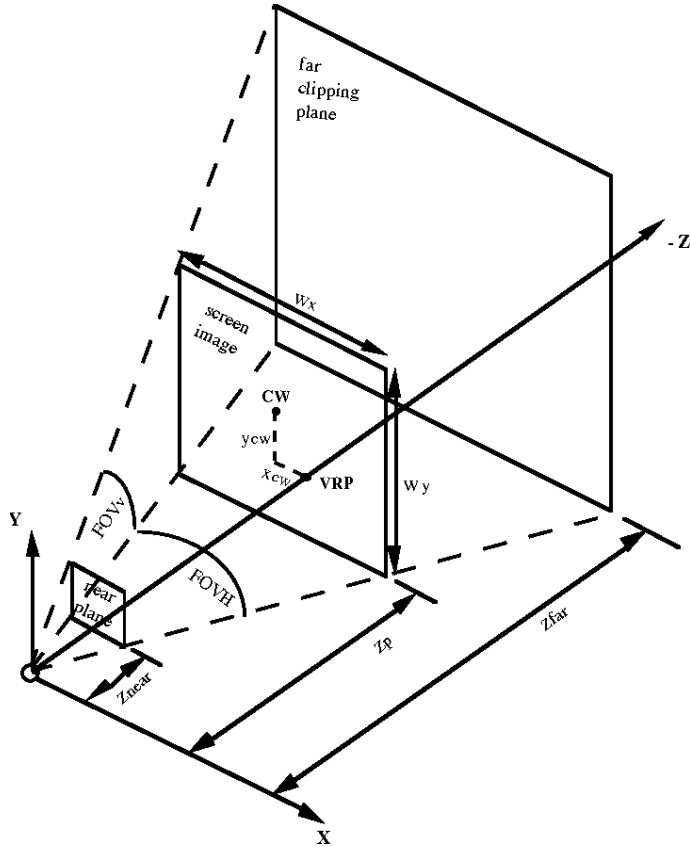
Graphical Display

- Personal displays
- Large volume displays
 - Active glasses
 - Workbenches;
 - Microsoft Surface
 - Caves;
 - Walls;





The Battle Front ...



An anchoring research article: The Visual Display Transformation for Virtual Reality by Warren Robinett and Richard Holloway <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.43.5727&rep=rep1&type=pdf>

What's well-understood?

- UNC VR Software (ref. the Anchoring Paper before)

- writing the visual display code for a virtual reality system so,
 - multiple users inhabit the same virtual world simultaneously;
 - each user has a stereoscopic display;
 - the user's viewpoint is measured by a head tracker;
 - the display code matches the geometry of the HMD, tracker, and optics;
 - various HMDs and trackers can be supported by changing parameters of the display code;
 - the user can fly through the world, tilt the world, and scale the world; and
 - the user can grab and move virtual objects.

Symbol	Coordinate Systems	Instances	Function	Static / Dynamic
T_{W_O}	Object to World	1 per object	position of object in world (changes when object moves)	dynamic
T_{R_W}	World to Room	1 per user	position of room in world (changes when flying, etc.)	dynamic
T_{TB_R}	Room to Tracker Base	1 per user	position of tracker in room	static
T_{HS_TB}	Tracker Base to Head Sensor	1 per user	measurement of head position and orientation by tracker	dynamic
T_{H_HS}	Head Sensor to Head	1 per user	position of sensor on HMD	static
T_{E_H}	Head to Eye	2 per user	positions of left and right eyes	static
T_{N_E}	Eye to Normalized	2 per user	off-center perspective projection	static
T_{US_N}	Normalized to Undistorted Screen	2 per user	convert to device coordinates	static
T_{S_US}	Undistorted Screen to Screen	2 per user	optical distortion correction	static

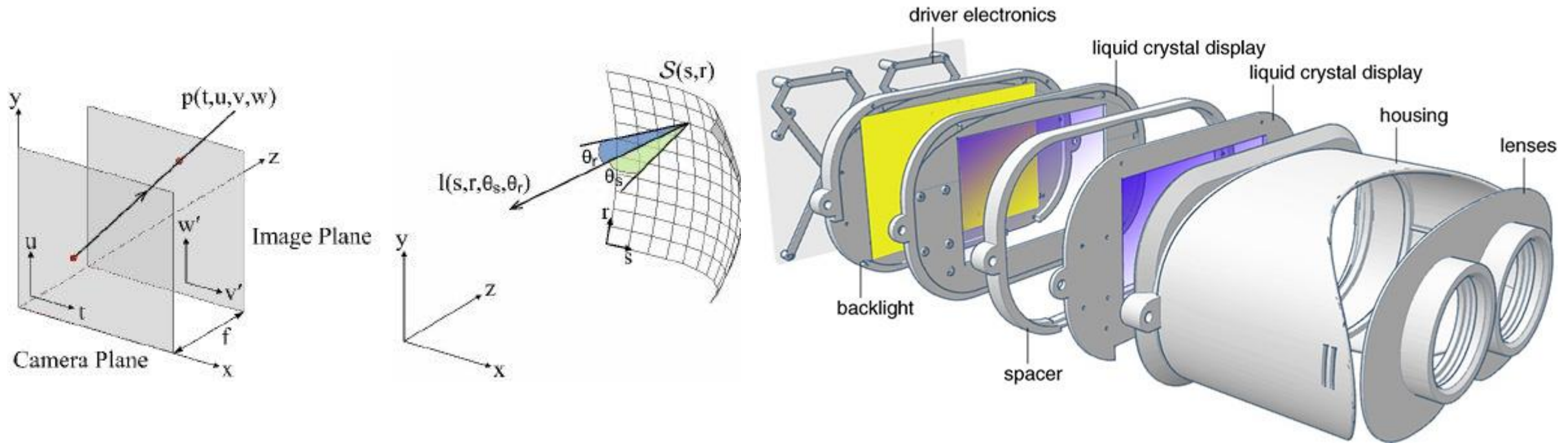
$$T_{S_O} = T_{S_US} T_{US_N} T_{N_E} T_{E_H} T_{H_HS} T_{HS_TB} T_{TB_R} T_{R_W} T_{W_O}$$

One Challenge - Recreating Visual Depth Cue

- Binocular Disparity (Stereopsis)
- Motion Parallax
- Binocular Occlusions
- Convergence/Divergence
- **Result - tolerable convincing illusion of depth**

- **What's missing?**
- Another depth cue: **focus** (imaging manually zooming in/out of an outdoor scene)

4-D Light Field Displays

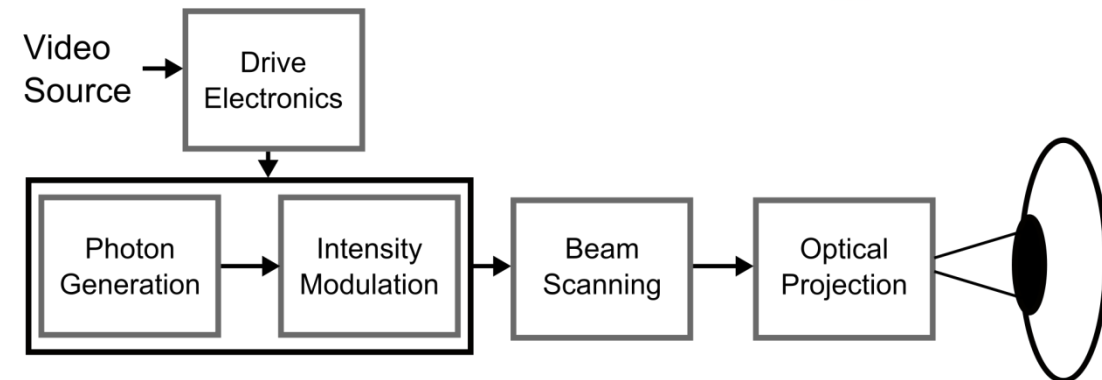
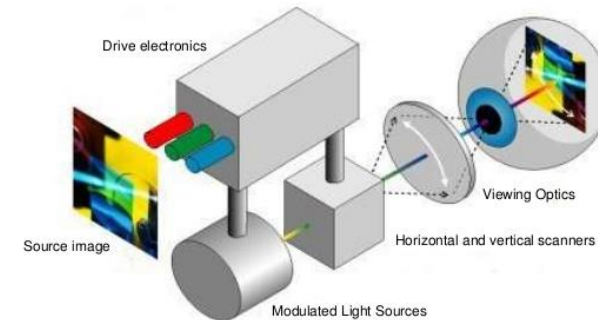


<https://www.researchgate.net/publication/259764186>

<https://spectrum.ieee.org/tech-talk/consumer-electronics/gaming/4d-light-field-displays-are-exactly-what-virtual-reality-needs>

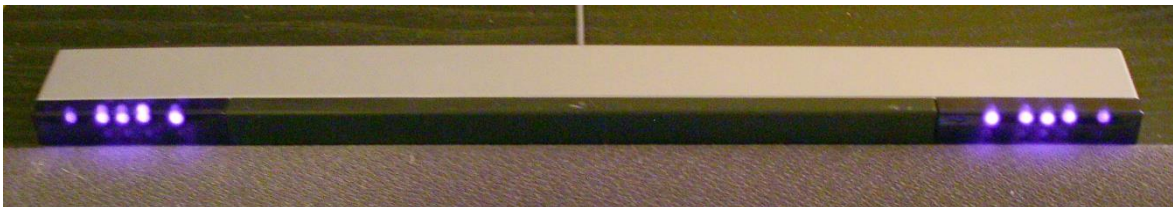
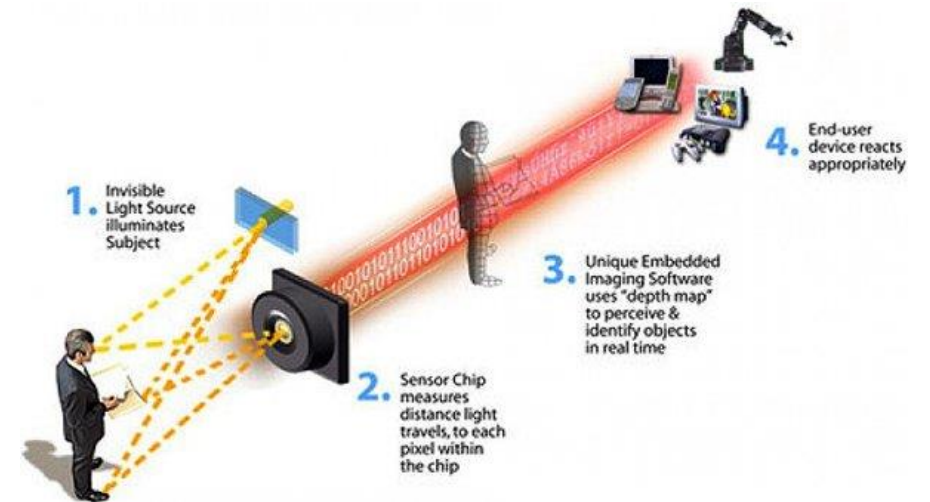
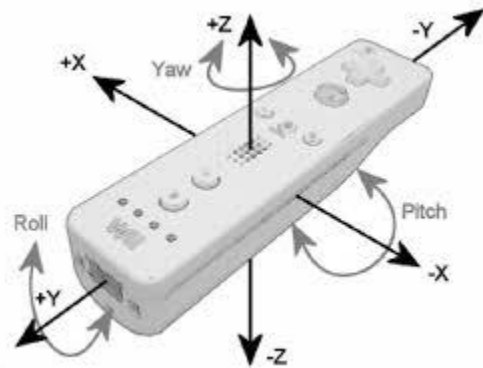
Personal Observations

- 1st Generation VR (Accessing human sensory channels, Passive)
- 2nd Generation VR (“Hacking” human sensory channels, Active, Automated, i.e. 4D Light Field Display, Structured Light)
- 3rd Generation VR (AI, Autonomous)
- Where are we now? Approx. 1.5?



Input Devices: Tracking Interfaces

- Measure head, body, hand or eye motion, ideally in 6-DoF
- Tracking technologies: *active*, *passive*, or *inertial*

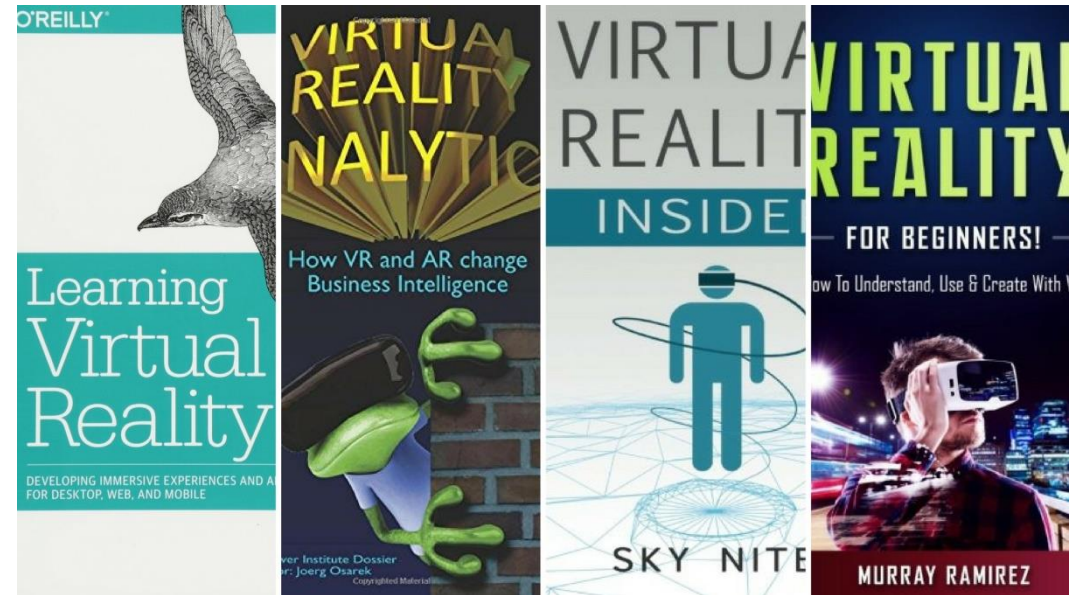


Technology Categorization

Trackers measure the motion of “objects” such as user’s wrist or head **vs.** a fixed system of coordinates.

Technologies to perform this task:

- Electromagnetic trackers;
- Ultrasonic trackers;
- Mechanical trackers;
- Inertial trackers;
- GPS
- **Vision-based trackers** (new and geared to be favorite)



Tracker Characteristics

An electronics/signal processing approach:

- Measurement rate – Readings/sec;
- Sensing latency;
- Sensor noise and drift;
- Measurement accuracy (errors);
- Measurement repeatability;
- Tethered or wireless;
- Work envelope;
- Sensing degradation



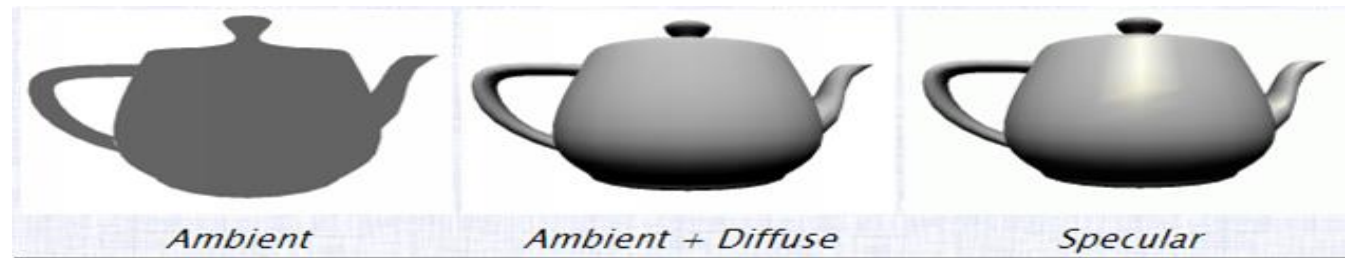
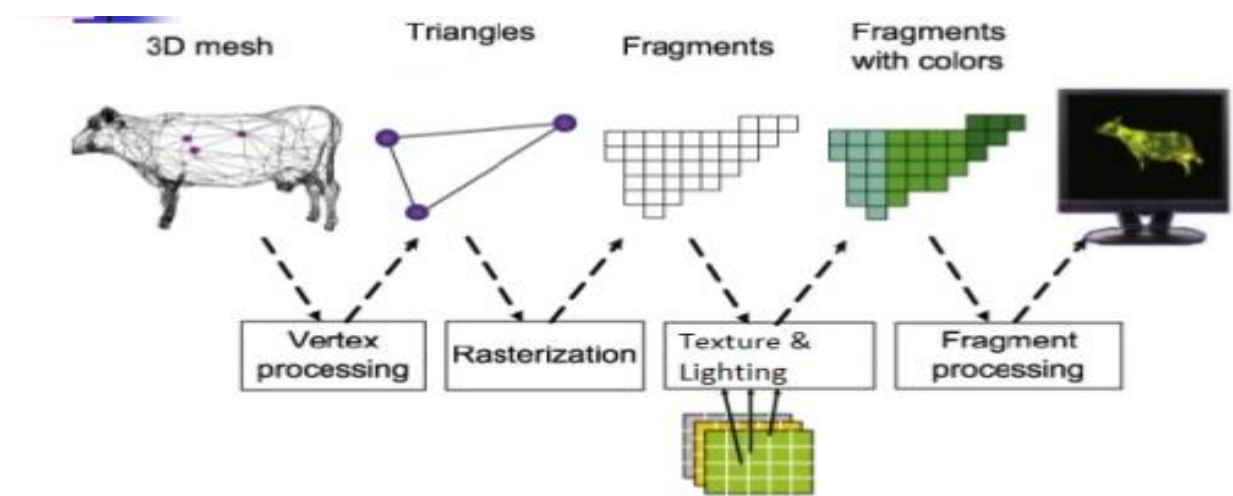
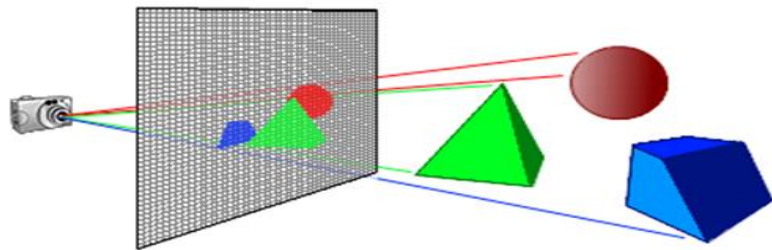
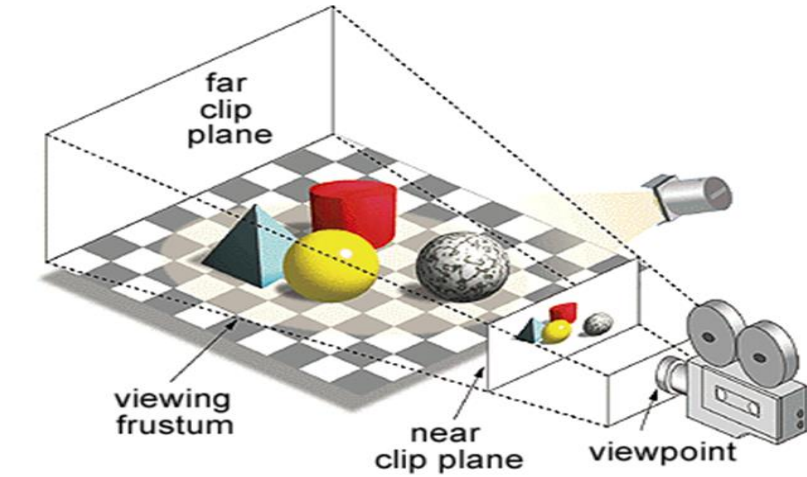
Application focused, i.e. Gesture Interaction

- To enable “Direct Manipulation” or “seamless User Interface”, interaction principles should be independent of any particular hardware. The focuses are on:
 - How does the user get content (both data and structure) into digital form? – i.e. Virtual World/Environment Builder, Virtual Object Modeller.
 - How does the user navigate around the content? – i.e. Virtual Environment Navigator.
 - How does the user manipulate the content (restructuring, revising, replacing)? – i.e. Animation/Simulation Engine.
- The **KEY** role of VR software...

VR Software

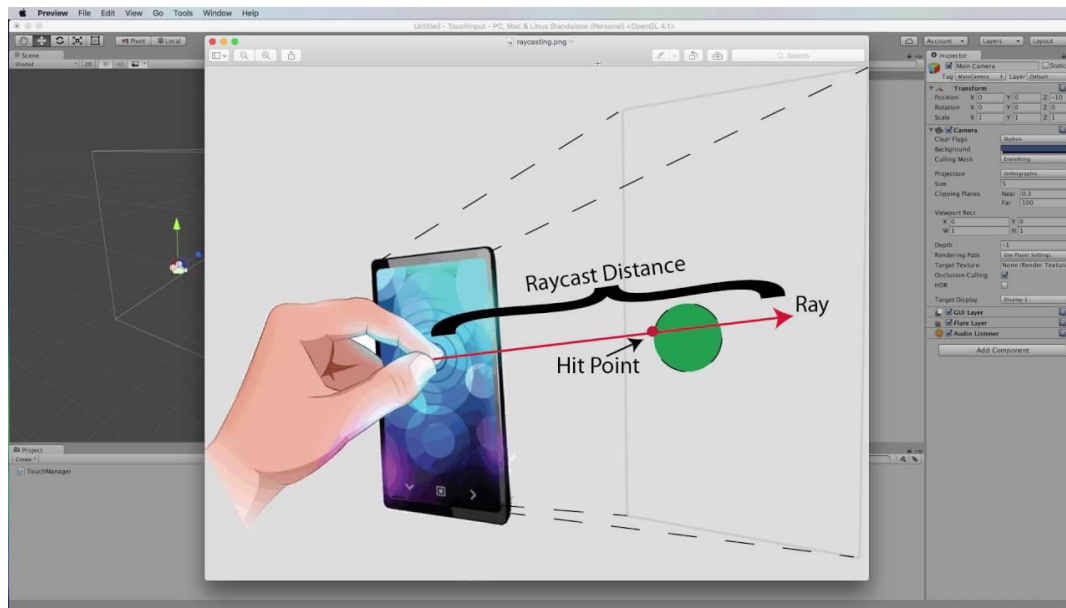
- ... not just device drivers but
- Virtual World (or Virtual Environment) Builder
 - The core of a VR system
 - Enables updates and rendering, interactions, object behaviors, simulations of physical laws, audios, and network issues, etc.
 - Commercial software, i.e., Unity, Unreal, 3D Studio Max, AutoCAD, Solidwork, CATIA, AVEVA, QT Modeller ...
 - Or, Graphics API-based, DirectX, OpenGL, etc.

The Heart - Real-time Rasterizer

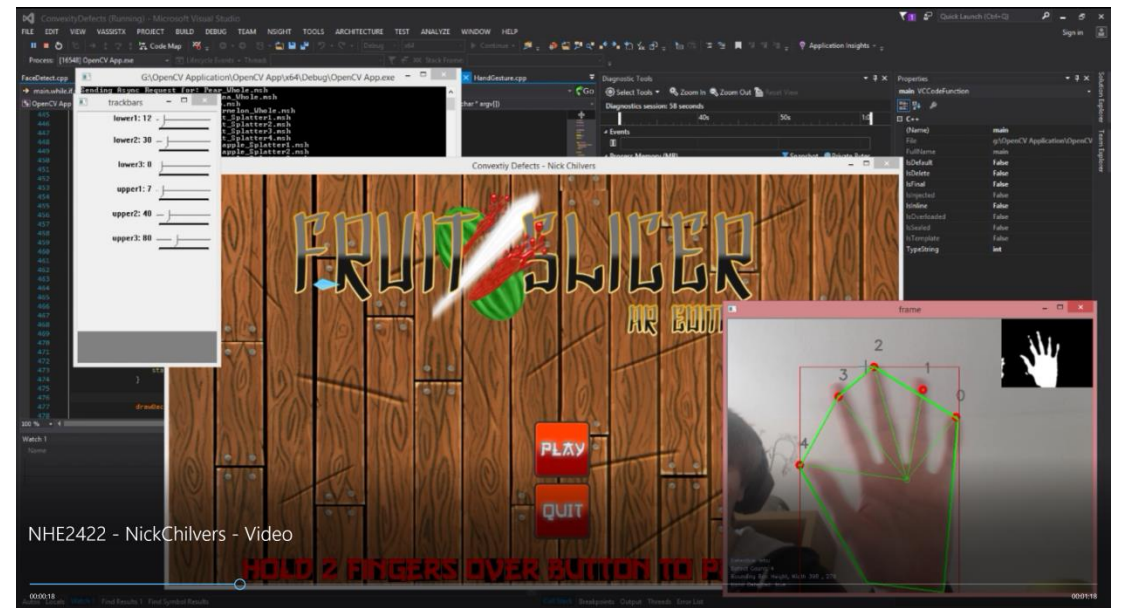


VR Software - Handles Interactions

- Pick Screen/Ray-casting

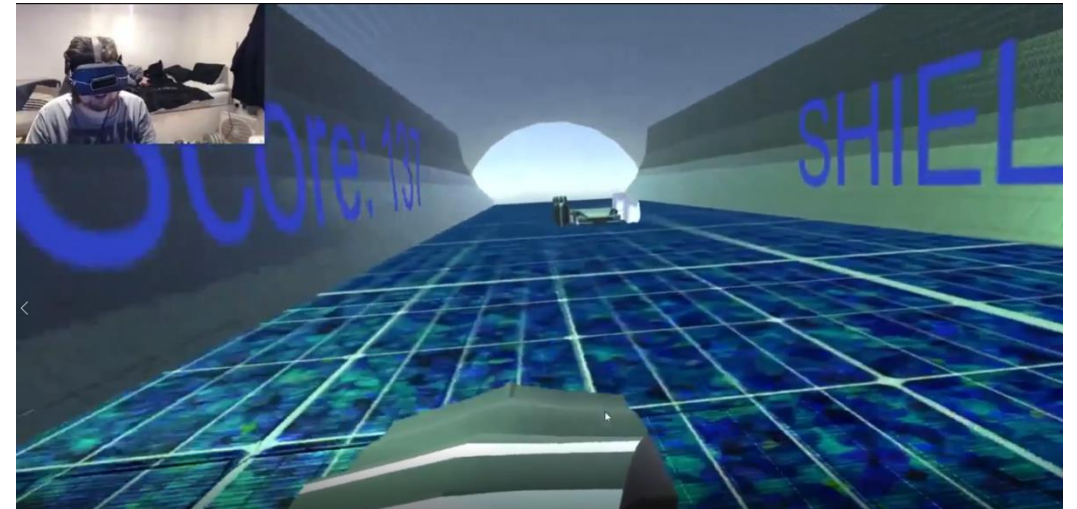


- Virtual-hand/Gesture Recognition



VR Software – Handles Navigation

- Steering mode
 - hand-directed
 - gaze-directed
 - physical devices (steering wheel, flight sticks)
- Target-focused
 - point at object, list of coordinates
- Path planning
 - place markers in world



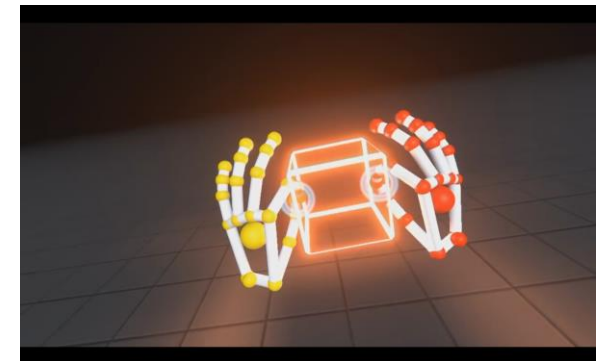
VR Software – Handles Simulation

- Physics-based activities
- Event-triggering mechanisms
 - Collision detection
 - Separating Planes
- Level-of-Details (LoD) Control
- Multimillion-particle simulations and volumetric processing
- SIGGRAPH is a gold mine



3. How to develop a Desk-top VR system? - Online Demos

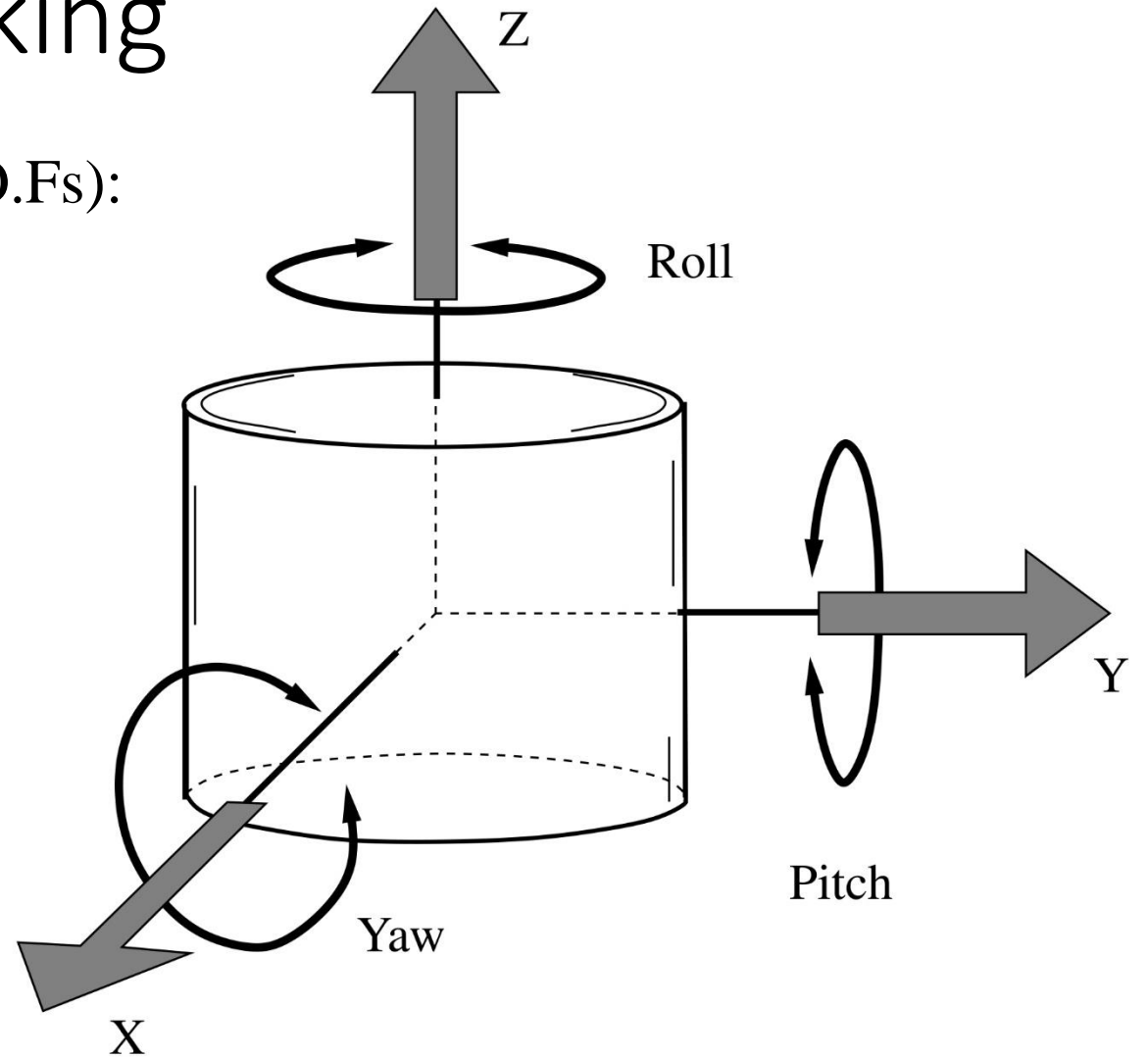
- <https://www.youtube.com/watch?v=Jd3-eiid-Uw&t=106s>
- https://www.youtube.com/watch?v=h9kPI7_vhAU
- <https://www.youtube.com/watch?v=rnlCGw-0R8g>



An Analogue of 6DoF Tracking

Virtual objects have 6 degrees of freedom (D.O.Fs):
three translations, three rotations.

- Capable of object manipulation.
- Still needing position information for full interaction.



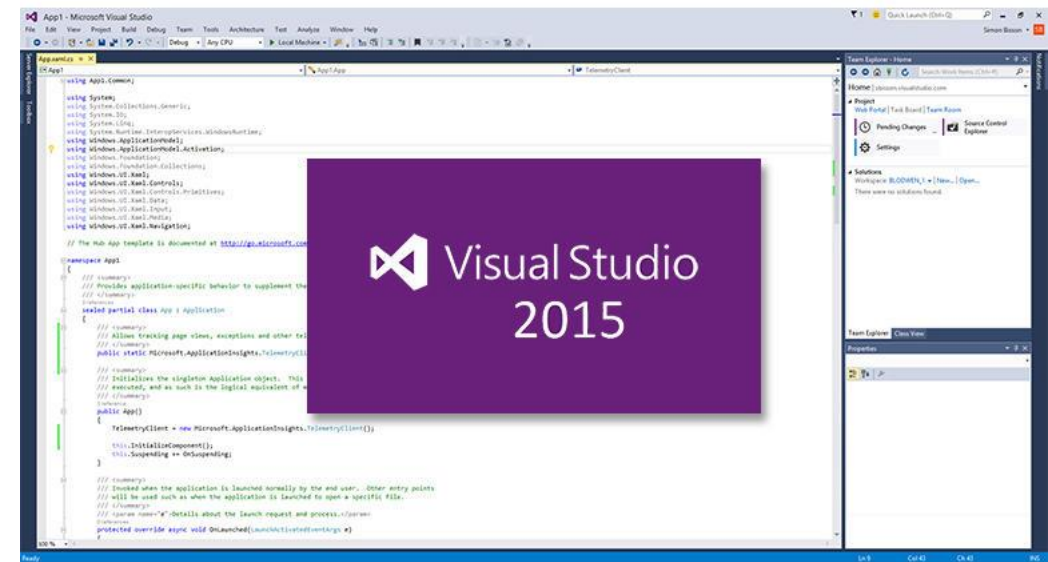
3-D System of coordinates of a VR object

Tutorial Goals

- A DIY VR Software Solution
- Accessing a real-time rendering pipeline
 - Model manipulation
 - Camera control/navigation
 - ~~Simulations~~
- through CV-based tracking

Development Tools

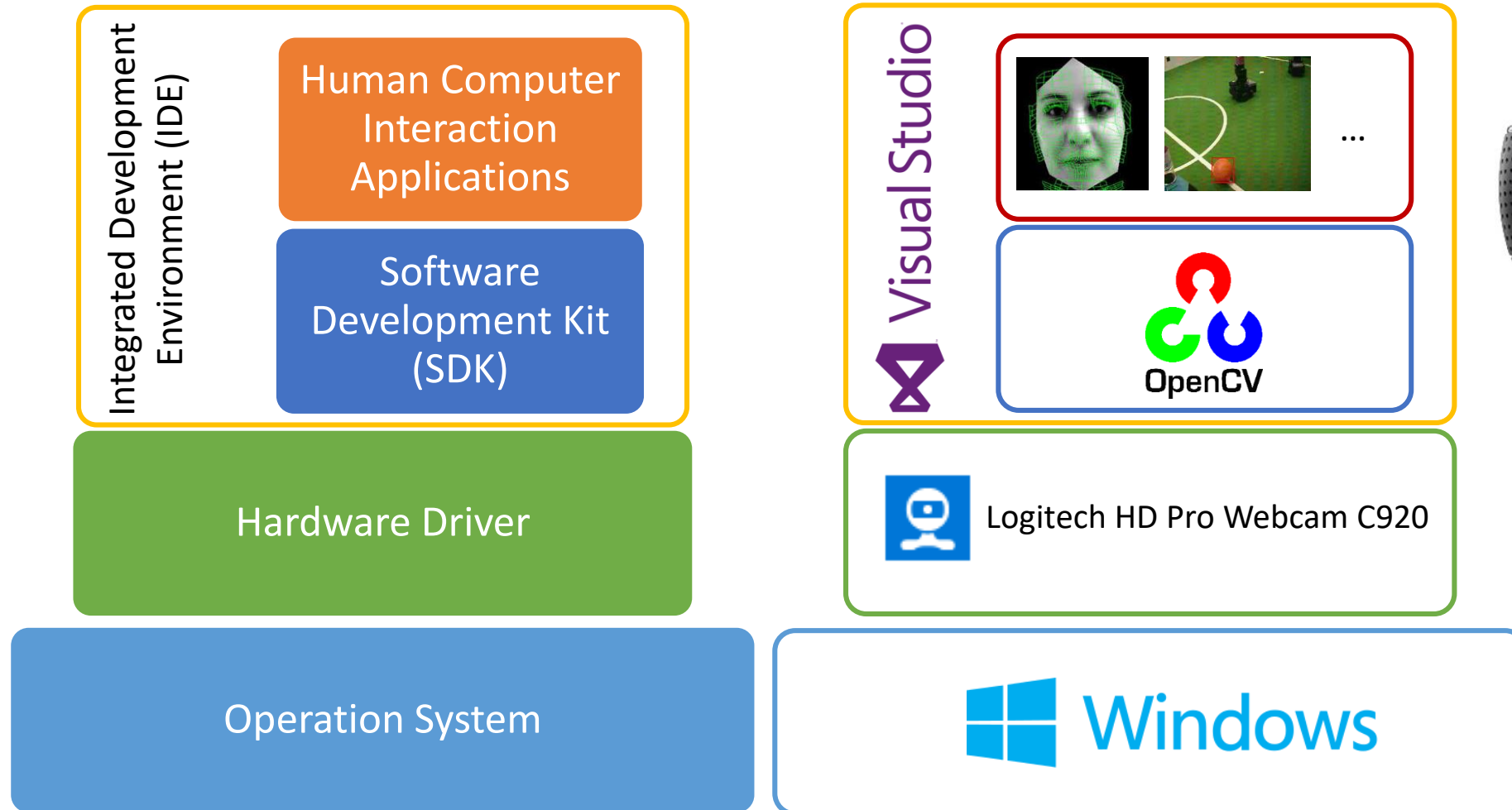
- Interactive Development Environments (IDE)
 - Visual Studio, Eclipse...
- Sophisticated environments for the development of software:
 - Intelligent Editors – think word processor for source code
 - Debuggers – allows you to watch the code execute to find problems (bugs)
 - Profilers – enables analysis of the efficiency of developed code



Development Tools

- Application Programming Interfaces (APIs)
 - Libraries of useful code to use within into your source code
 - **DirectX** – graphics/audio
 - OpenGL – graphics
 - Open Audio – audio
- Middleware – useful software APIs that facilitate various smaller tasks in games (goes between other components)
 - Physics, Data Processing, Networking, AI, User Interfaces
 - Goto: <http://www.gamemiddleware.org/middleware/index.html>

Webcam Projects Development Framework



Where everything starts

- Image Pixels



video



frames

One frame = image



image

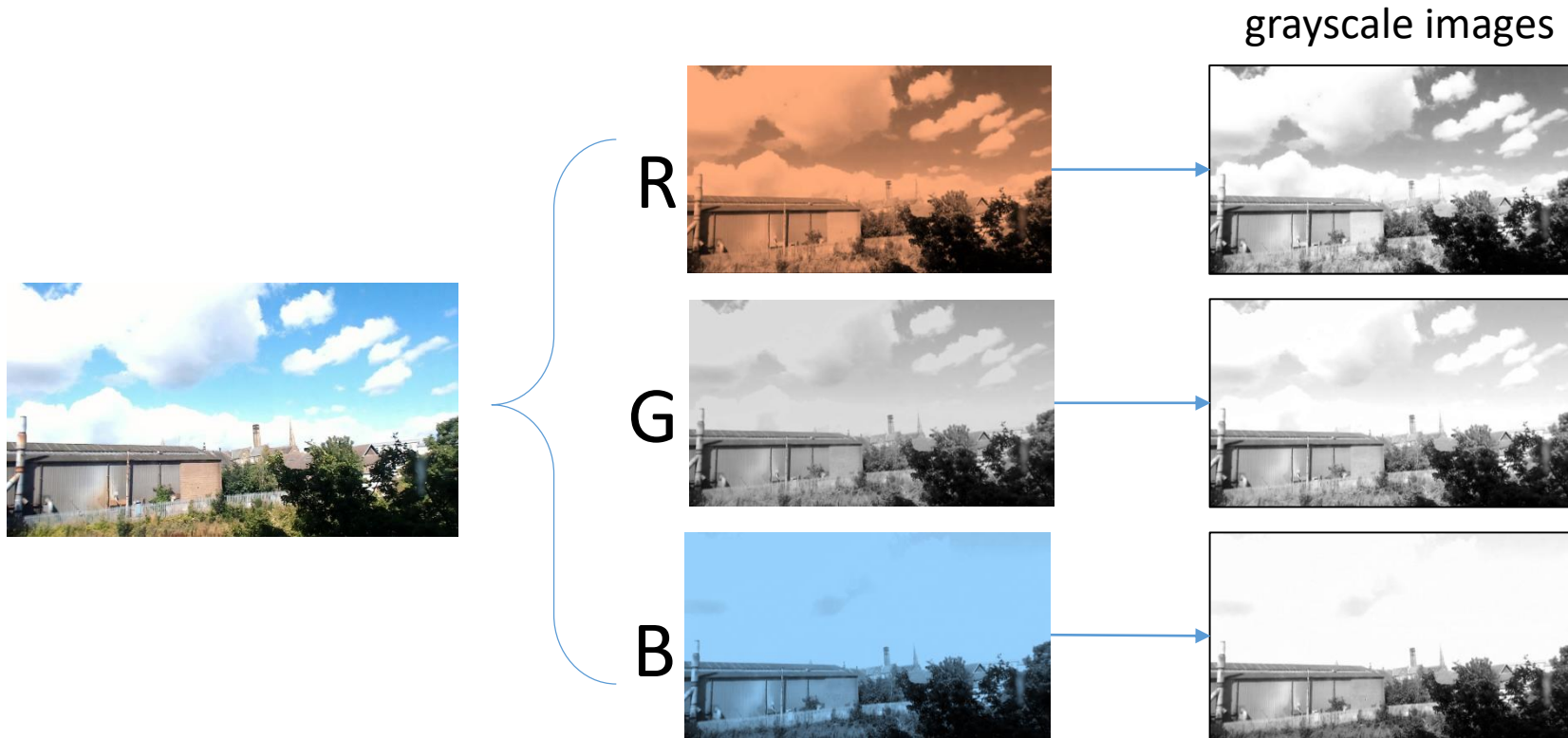
Pixel matrix



One pixel

Understanding image channels

- Colour digital images are made of pixels, and pixels are made of combinations of primary colours. A channel in this context is the grayscale image of the same size as a colour image, made of just one of these primary colours.
- For instance, an image from a standard digital camera will have a red(R), green(G) and blue(B) channel.
- A grayscale image has just one channel.



Tutorial 1: Hands-on Development for Hands-off VR System - Preparation

- Setup a C++ OpenCV Project in Visual Studio
- Load an image from a hard disk
- Display an image
- Use webcam to capture an image
- Operate on image sequences
- Simple video processing



Introducing OpenCV

- OpenCV (Open Source Computer Vision Library) is an open source computer vision and machine learning software library.
- The library has more than 2500 optimized algorithms, which includes a comprehensive set of both classic and state-of-the-art computer vision and machine learning algorithms.
- It has C++, C, Python, Java and MATLAB interfaces and supports Windows, Linux, Android and Mac OS.

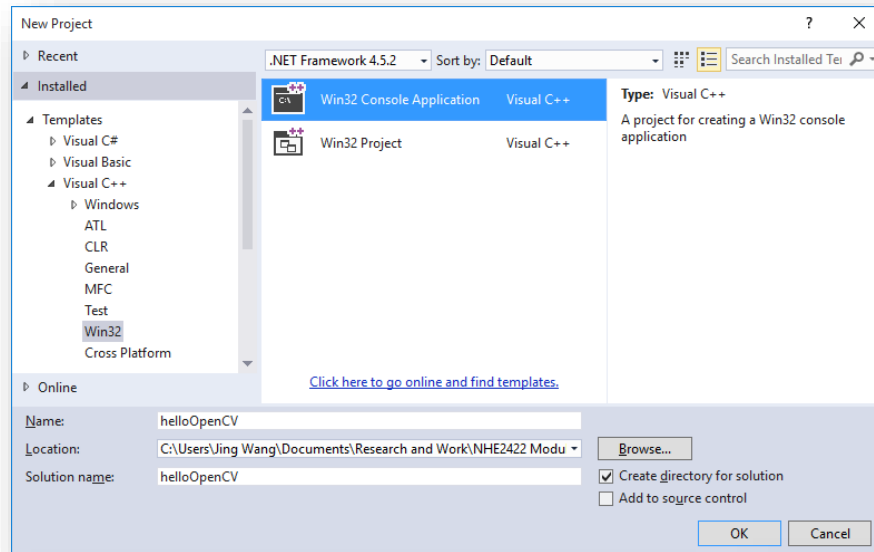
Official Website

<http://opencv.org/>

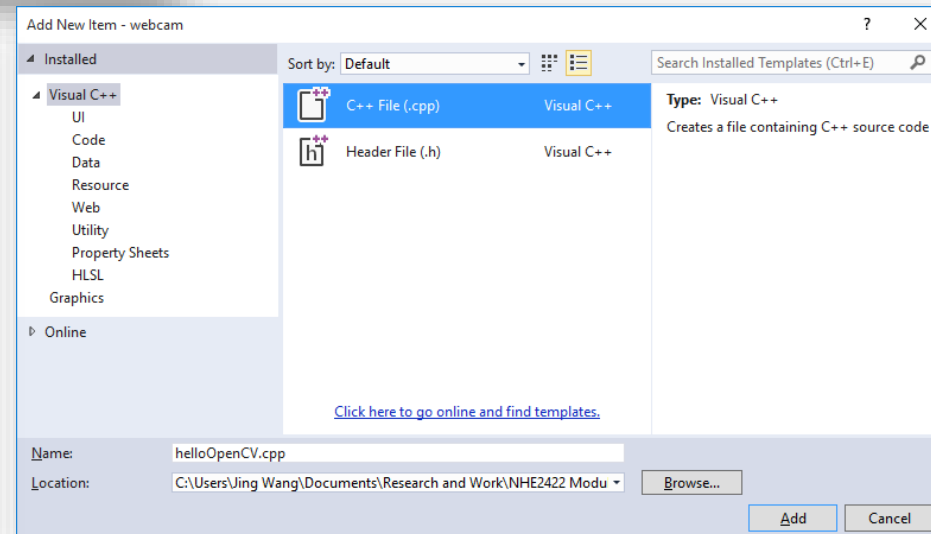
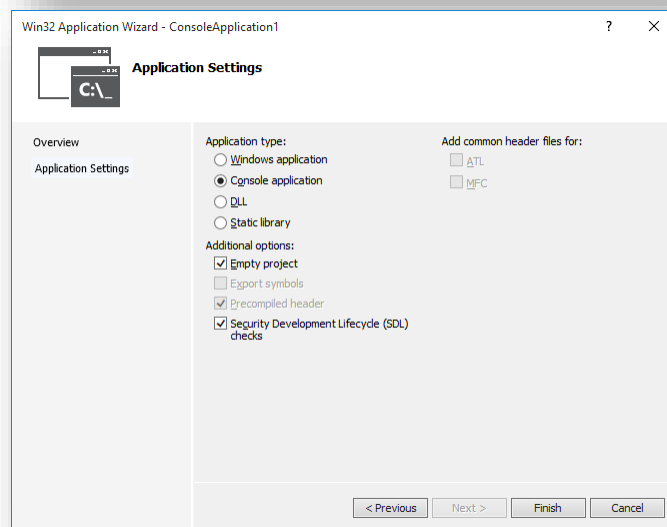
Programming Tutorials

http://docs.opencv.org/master/d9/df8/tutorial_root.html

Step1: Creating an OpenCV Project

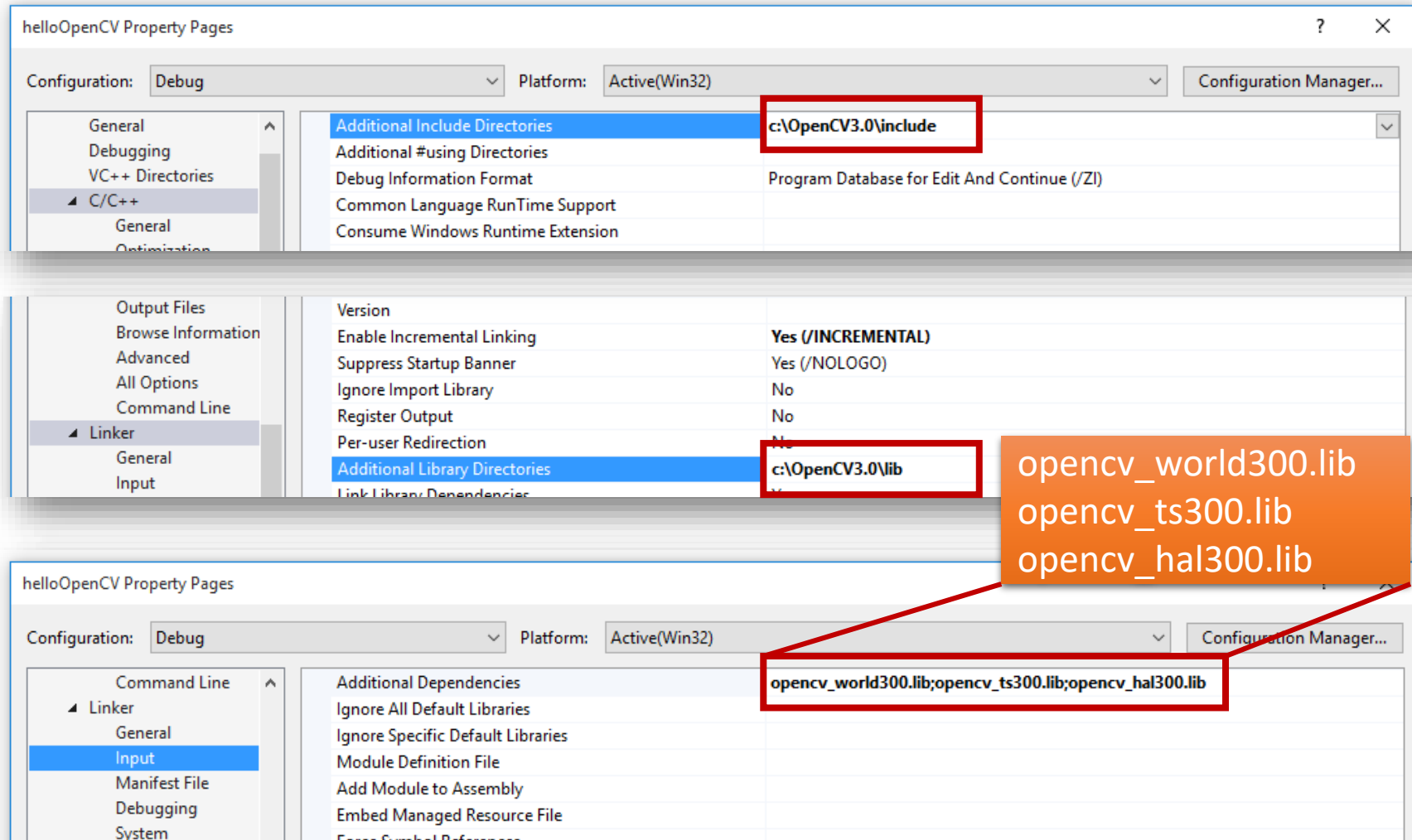


1. Creating a new win32 C++ console application
2. Name the project
3. Use Application Wizard to choose “Console application” and an empty project
4. Add a new source files item named as “HelloOpenCV.cpp”



Step1: Creating an OpenCV Project

Before coding, it is important to tell Visual C++ where to find the libraries and includes files



Step 1: Creating an OpenCV Project

- When using openCV, following head files are usually included:

```
#include "opencv2\imgproc\imgproc.hpp"  
#include "opencv2\highgui\highgui.hpp"  
#include "opencv2\opencv.hpp"
```

- All classes and functions are defined within the same space cv

Step 2: Loading and displaying image

The first thing to do is to declare a variable that will hold the image. Under OpenCV3.0, you can define an object of class `cv::Mat`

```
cv::Mat image;
```

The image can be loaded by using reading function `cv::imread()`, which can visit the image from file, decode it, and allocate the memory. You can use absolute path and relative path to locate an image file stored on your local hard disk:

```
image = cv::imread("dogs.png");
```

You can also check if the image has been correctly read by using a `cv::Mat` member variable "data":

```
if (!image.data) {  
    //process the error that no image has been created..  
}
```

Step 2: Loading and displaying image

To display the image, a `highgui` module is provided by openCV, The name of the image displaying window should be specified:

```
cv::imshow("Image Window", image);
```

Since it is a console windows that will terminate at the end of the main function, we add an extra `highgui` method to wait for a user key before ending the program

```
cv::waitKey(0);
```

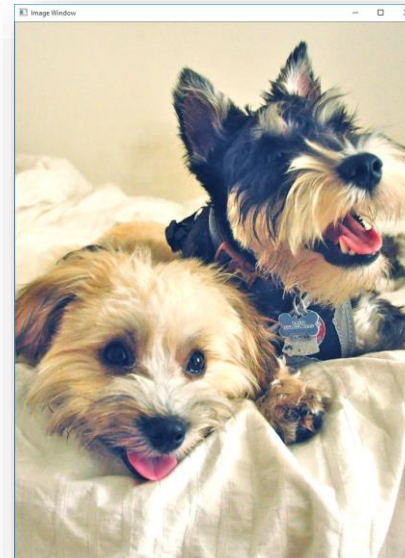
Step 3: Showing the result

After building the vc++ project, you can debug the code. The result is

```
#include "stdafx.h"

#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\opencv.hpp>

int main()
{
    cv::Mat image;
    image = cv::imread("dogs.png");
    cv::imshow("Image Window", image);
    cv::waitKey(0);
    return 0;
}
```

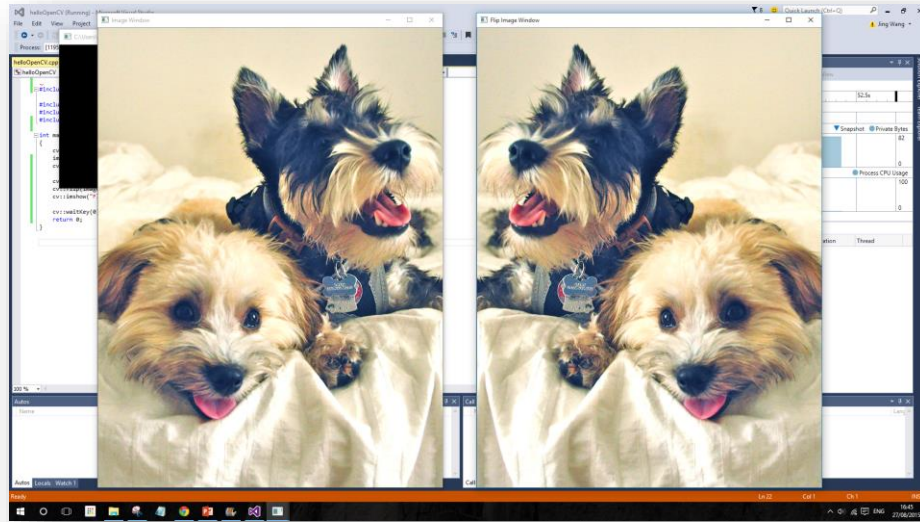


The program can be stopped by pressing any key when the "Image Window" is activated.

More...

OpenCV offers a wide selection of processing functions. Most of them are easy to use. For example, The image can be flipped horizontally by adding following code:

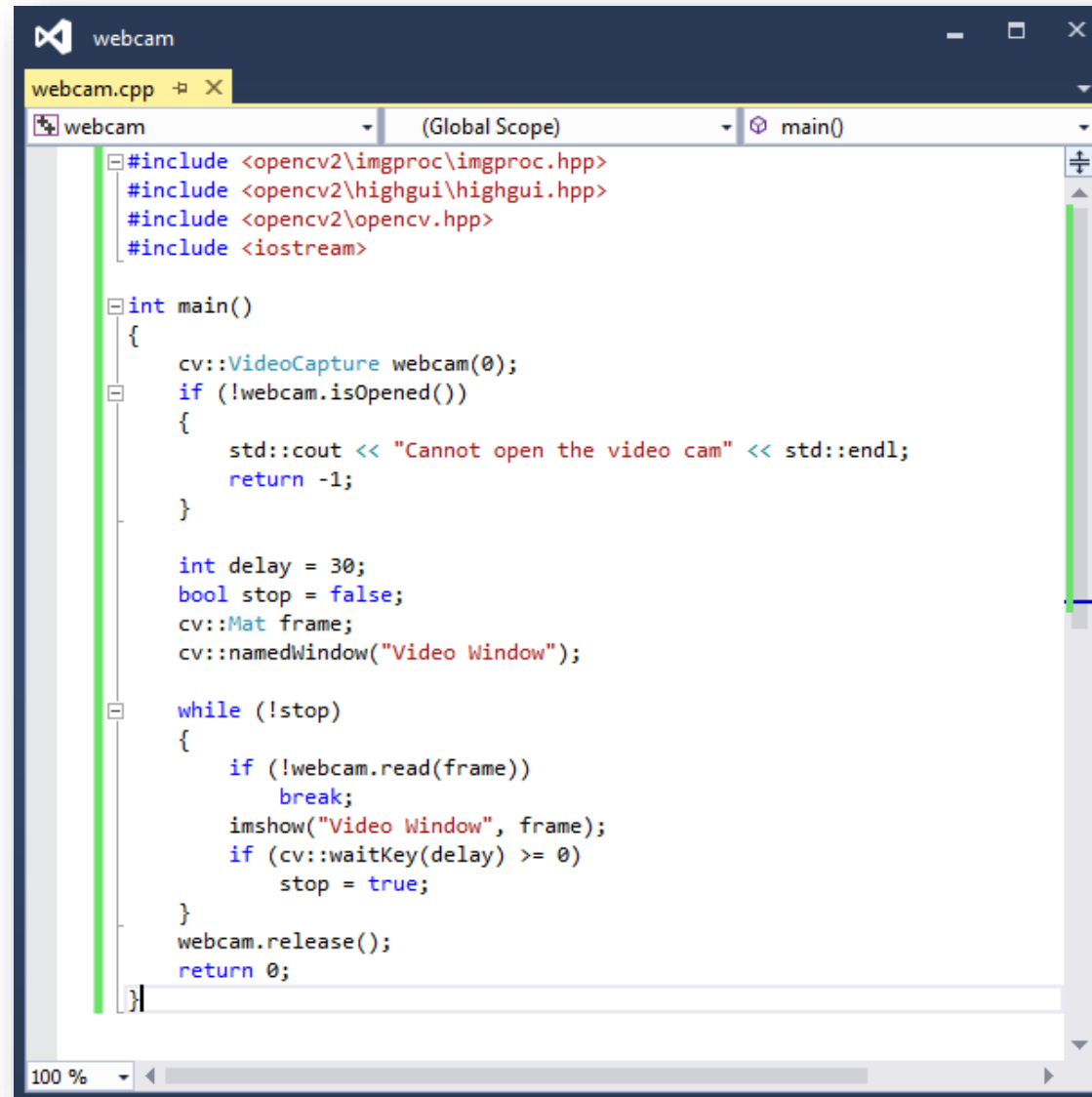
```
cv::Mat flipImage;  
cv::flip(image, flipImage, 1);  
cv::imshow("Flip Image Window", flipImage);
```



You can also save the processed image on hard disk:

```
cv::imwrite("flipped.bmp", flipImage);
```


Step 4: Reading webcam video streams



```
webcam
webcam.cpp
webcam (Global Scope) main()
#include <opencv2\imgproc\imgproc.hpp>
#include <opencv2\highgui\highgui.hpp>
#include <opencv2\opencv.hpp>
#include <iostream>

int main()
{
    cv::VideoCapture webcam(0);
    if (!webcam.isOpened())
    {
        std::cout << "Cannot open the video cam" << std::endl;
        return -1;
    }

    int delay = 30;
    bool stop = false;
    cv::Mat frame;
    cv::namedWindow("Video Window");

    while (!stop)
    {
        if (!webcam.read(frame))
            break;
        imshow("Video Window", frame);
        if (cv::waitKey(delay) >= 0)
            stop = true;
    }
    webcam.release();
    return 0;
}
```

How it works...

To process a video sequence, we need to be able to read each of its frames, OpenCV place an object of class `cv::VideoCapture` to perform frame extraction from video files and webcams.

```
cv::VideoCapture object_name(parameters);
```

For using video files, *parameters* is the path of the file. For example,

```
cv::VideoCapture videoFile("../bike.avi");
```

For using webcams, *parameters* is an index number of each camera. "0" means default camera. For example,

```
cv::VideoCapture webcam(0);
```

If you have more webcams installed, the index number can be 0,1,2,...

How it works...

Once the `cv::VideoCapture` object has been created, it can be verified through the `isOpened` method:

```
if (!webcam.isOpened())
{
    std::cout << "Cannot open the video cam" << std::endl;
    return -1;
}
```

For accessing a frame, a method “`read()`” of `cv::VideoCapture` object can be used. A variable also need to be defined for storing that frame.

```
cv::Mat frame;
webcam.read(frame);
```

If the “`read()`” failed to get a frame, it will return a “0”. In this case, the verification and frame capture can be coded in one line:

```
if (!webcam.read(frame))
    \\ do something
```

How it works...

while loop is used for renewing each frame. In this case, the “read()” should be placed inside the loop. For playing the current frame in an image window, some parameters are required to control the loop.

A flag should be defined for ending the loop when a key is pressed:

```
bool stop = false;
int const delay = 30;
while (!stop)
{
    if (!webcam.read(frame))
        break;
    imshow("Video Window", frame);
    if (cv::waitKey(delay) >= 0)
        stop = true;
}
```

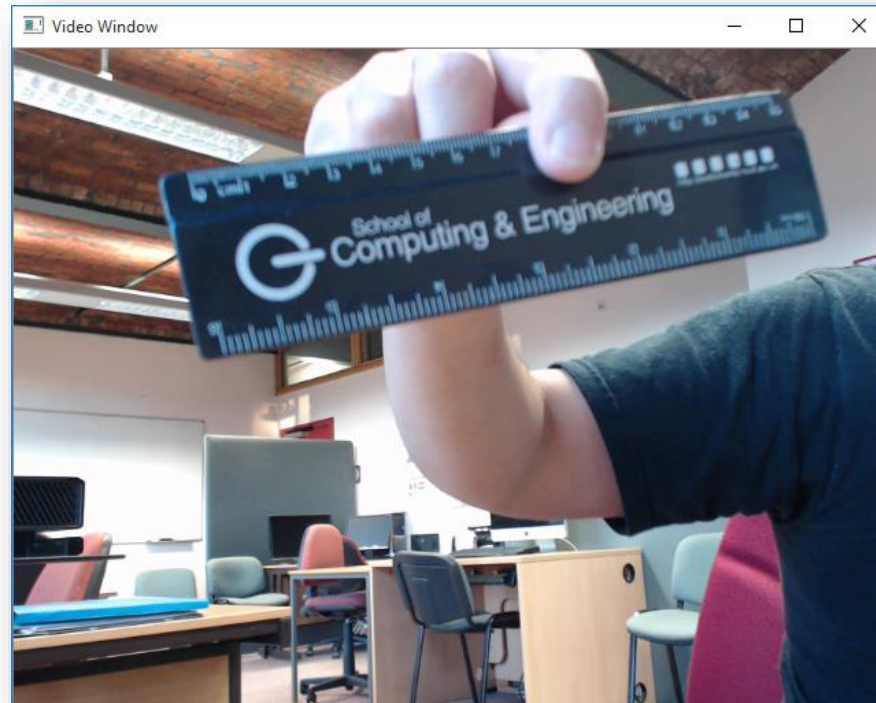
`cv::waitKey(delay)` waits for a pressed key and waits the event in milliseconds. 0 is the special value that means waiting a key to be pressed "forever". It returns the code of the pressed key or -1 if no key was pressed before the specified time had elapsed.

How it works...

The final statement calls the `release` method which will close the webcam.

```
webcam.release();
```

Running the programme



Step 5: Video Processing

Real-time video processing steps are usually added after getting each new frame. For example, we can change the colour images into grayscale images. We use the `cvtColor` function:

```
void cv::cvtColor(cv::Mat input, cv::Mat output, int method);
```

The function should be placed inside the loop:

```
while (!stop)
{
    if (!webcam.read(frame))
        break;
    imshow("Video Window", frame);
    cv::cvtColor(frame, output, CV_RGB2GRAY);
    imshow("Processed video", output);
    if (cv::waitKey(delay) >= 0)
        stop = true;
}
```

Step 5: Video Processing

The screenshot displays the Microsoft Visual Studio IDE with a C++ program for video processing. The code in the main window is as follows:

```
#include <opencv2/imgproc/imgproc.hpp>
#include <opencv2/highgui/highgui.hpp>
#include <opencv2/opencv.hpp>
#include <iostream>

int main()
{
    cv::VideoCapture webcam(0);

    if (!webcam.isOpened())
    {
        std::cout << "Cannot open the video cam" << std::endl;
        return -1;
    }

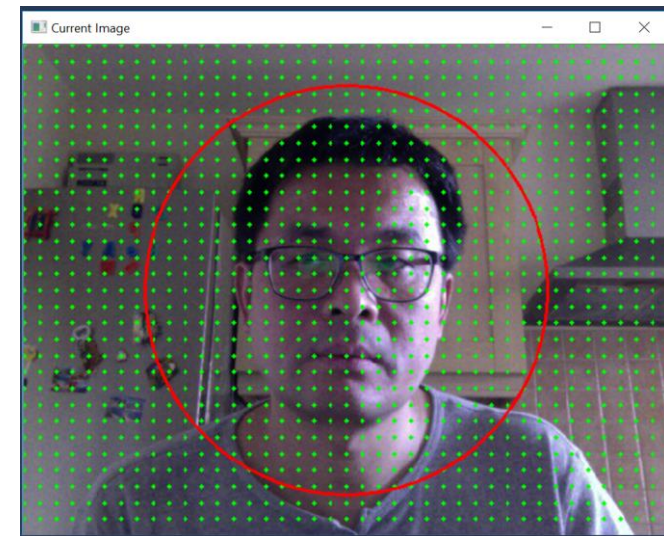
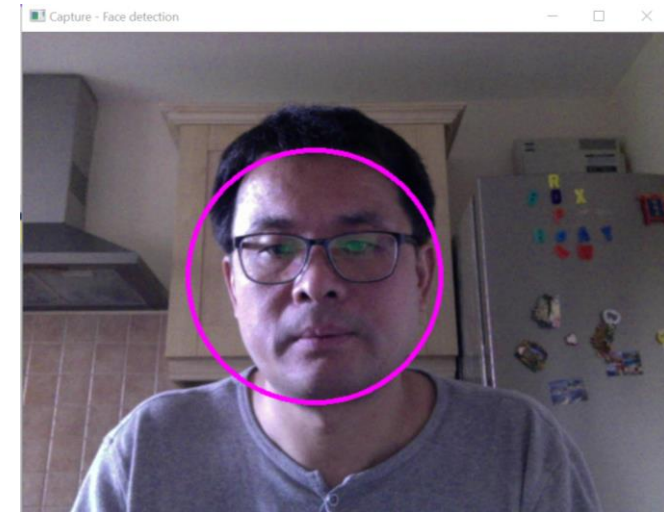
    int const delay = 30;
    bool stop = false;
    cv::Mat frame;
    cv::namedWindow("Video Window");
    cv::Mat output;
    while (!stop)
    {
        if (!webcam.read(frame))
            break;
        imshow("Video Window", frame);
        cv::cvtColor(frame, output, CV_RGB2GRAY);
        imshow("Processed video", output);
        if (cv::waitKey(delay) >= 0)
            stop = true;
    }
    webcam.release();
    return 0;
}
```

The interface also shows a 'Video Window' displaying a color frame of a person holding a book, and a 'Processed video' window displaying the same frame in grayscale. Diagnostic tools on the right show a 35-second session with CPU utilization and process memory usage. The system tray at the bottom indicates the date and time as 21:25 on 27/08/2015.

Tutorial 2: Handles Interactions and Navigation

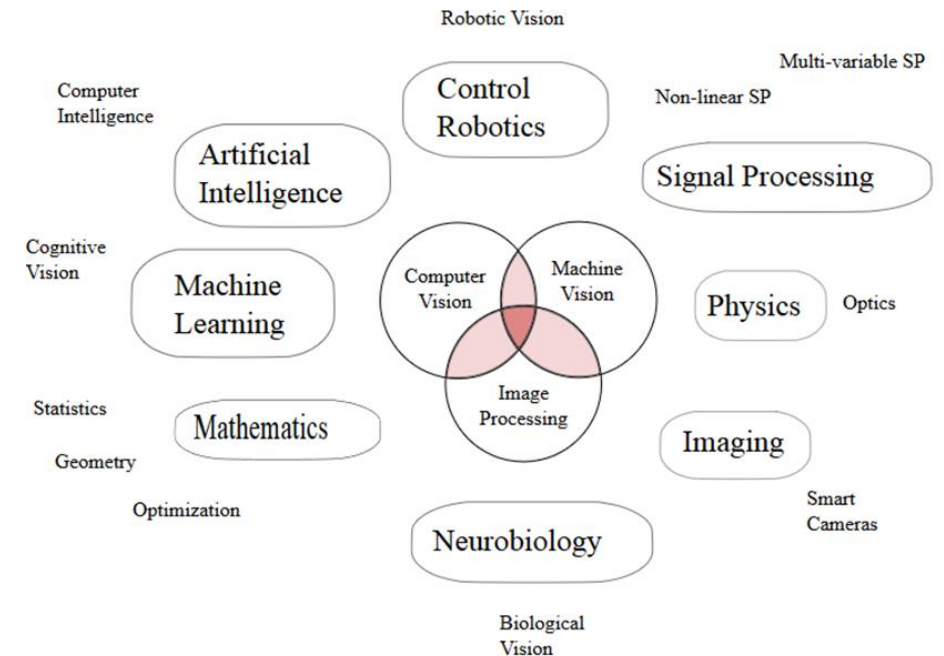
- Histogram concepts
- HSV Histogram building
- Histogram-based template matching
- Template matching-based “tracking”
- Access virtual object’s “WORLD” matrix
- Access camera’s “VIEW” matrix

Conventional Tracking (3D to 3D) vs. CV Tracking (2D to 3D)



Main CV Tasks

- Feature Engineering
- Image Stitching and Stereo Vision
- SfM and VSLAM
- 3-D Point Cloud Processing
- ...
- **Used for VR Input**
- Object Detection and Recognition
- Object Tracking and Motion Estimation
- ...



Solutions

- Easy cases:

- Stereopsis – 2 cameras generating a disparity map

$$[X Y Z W]^T = Q * [x y \text{ disparity}(x,y) 1]^T$$

- Hard cases:

- Converting 2D Image Coordinates to 3D World Coordinates
- Simplified way: $Z = 0$
- Other improvised methods

translate

$$\begin{aligned} x_{\text{camera}} &= x_{\text{world}} - \text{cameraX} \\ y_{\text{camera}} &= y_{\text{world}} - \text{cameraY} \\ z_{\text{camera}} &= z_{\text{world}} - \text{cameraZ} \end{aligned}$$

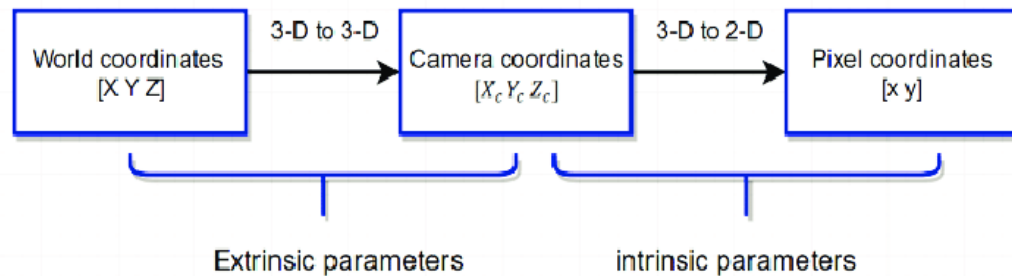
project

$$\begin{aligned} x_{\text{proj}} &= x_{\text{camera}} * d/z_{\text{camera}} \\ y_{\text{proj}} &= y_{\text{camera}} * d/z_{\text{camera}} \end{aligned}$$

scale

$$\begin{aligned} x_{\text{screen}} &= (w/2) + (w/2)*x_{\text{proj}} \\ y_{\text{screen}} &= (h/2) - (h/2)*y_{\text{proj}} \end{aligned}$$

... where (w,h) = canvas width and height



$$\lambda \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix} = \begin{bmatrix} f_x & 0 & x_{c0} \\ 0 & f_y & y_{c0} \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix} \begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$$

degree of freedom (for the depth) λ
 Point in Captured Image $\begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}$
 focal length f , center of the image $c0$
 rotational component r translate components t
 Point in Marker Coordinate $\begin{bmatrix} X_w \\ Y_w \\ Z_w \\ 1 \end{bmatrix}$
 Point in Camera Coordinate $\begin{bmatrix} r_{11} & r_{12} & r_{13} & t_x \\ r_{21} & r_{22} & r_{23} & t_y \\ r_{31} & r_{32} & r_{33} & t_z \end{bmatrix}$
 Point in Captured Image coordinate $\lambda \begin{bmatrix} x_c \\ y_c \\ 1 \end{bmatrix}$

Main Process Flow

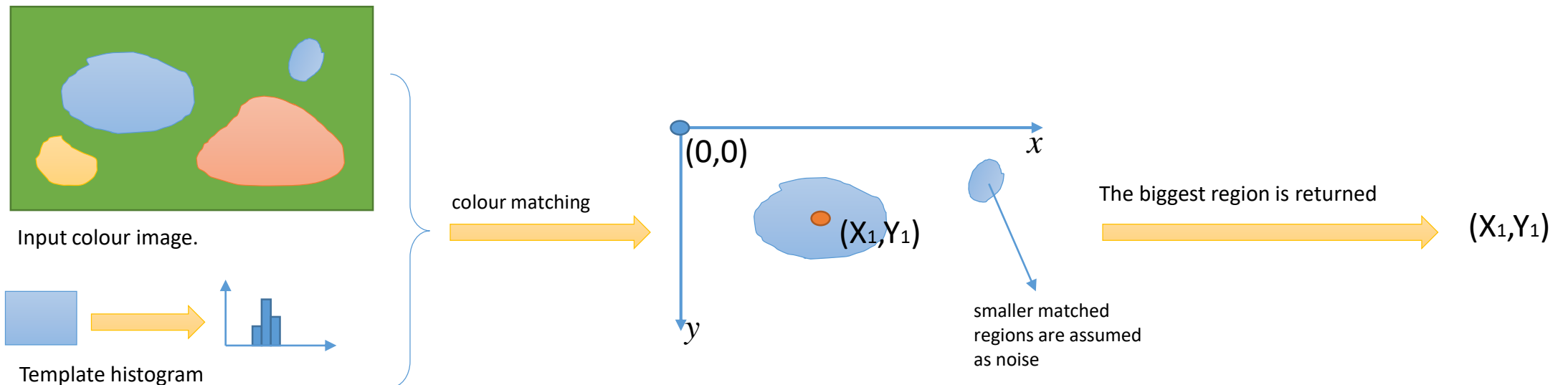
Using a colour histogram template to match image regions. The function returns the location of the biggest matched region.

Point colourTracking(const Mat &frame, const MatND &hsv_hist)

Return a cv::Point type. The type contains colour tracking result presented by (x,y) pairs.

Input colour image.

Input HSVhistogram is used as a colour template, which tells the function which colour should be tracked.

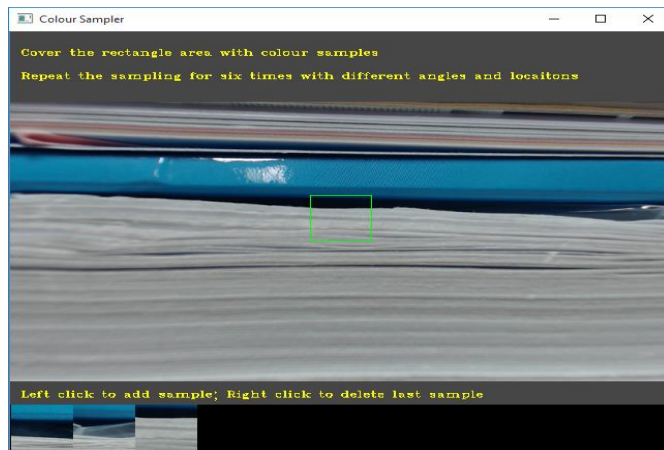


How to construct the histogram template?



Run the “HS_colourSensor.exe” to create a template histogram file first and then use this file with the main tracking function: colourTracking()

HS_colourSensor.exe

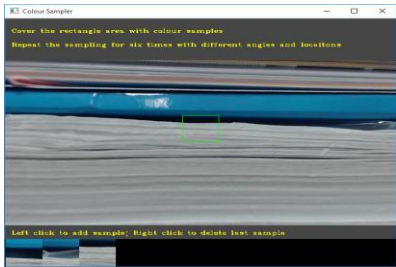


How to read the histogram file:

```
cv::MatND hist;
cv::FileStorage fs("colour_hist.yml", cv::FileStorage::READ);
if (!fs.isOpened()) {cout << "unable to open file storage!" << endl;}
fs["histogram"] >> hist;
fs.release();
```

The template file is saved as a “colour_hist.yml” file in the project folder. Read/load this file into cv::MatND as a variable, which is used for saving multidimensional matrices.

“Tracking” at Runtime



Colour Sensor (app)

save →



colour_hist.yml

use it in
your application →



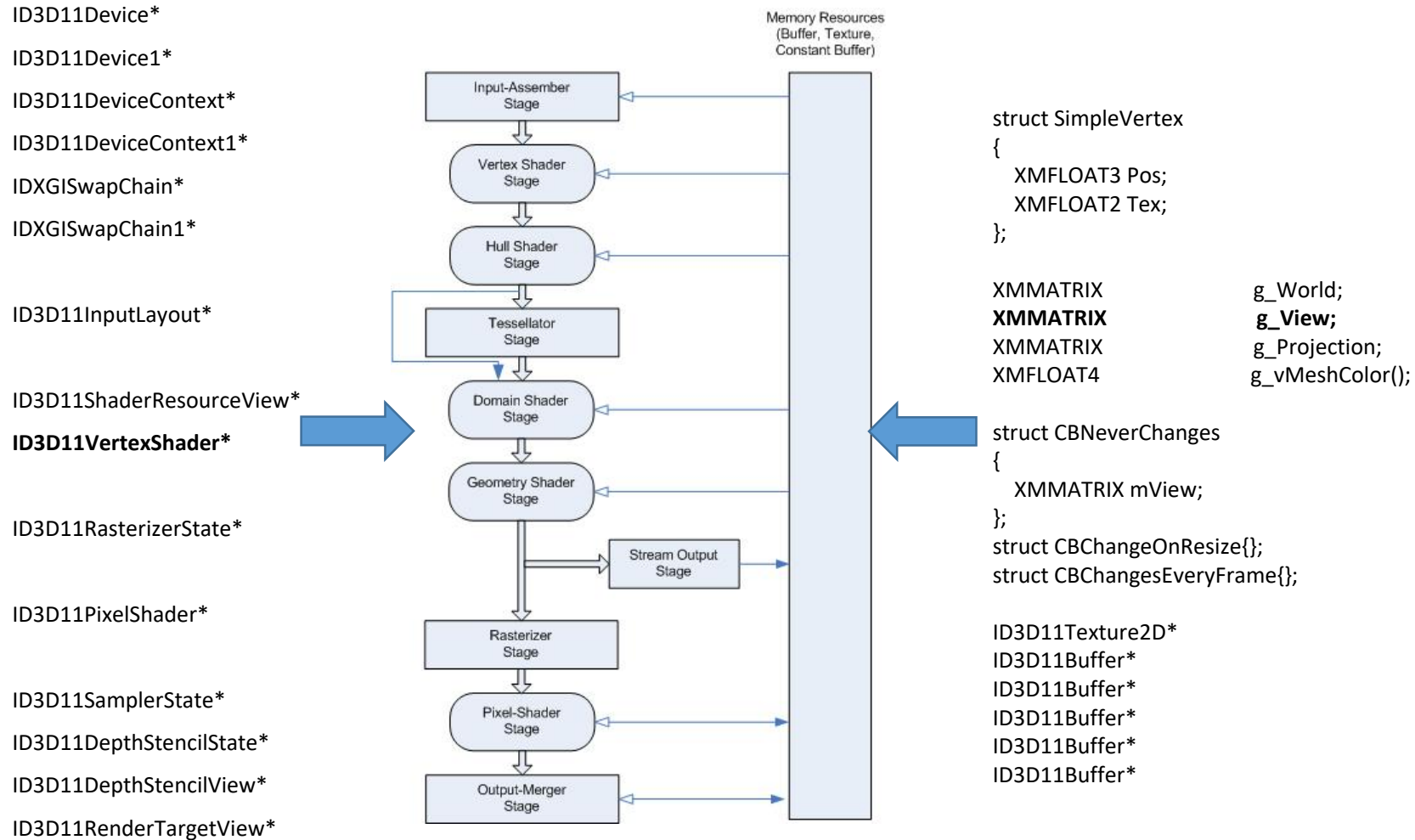
colourTracking()

(x,y)

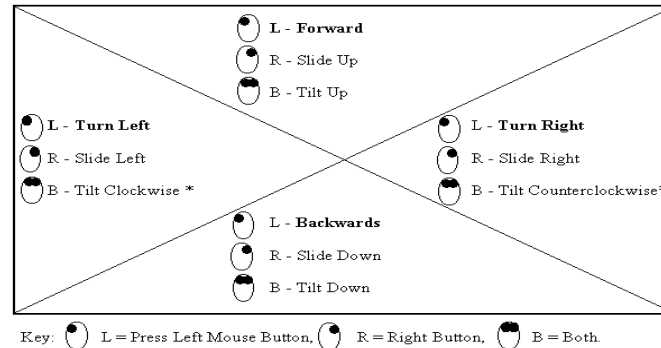
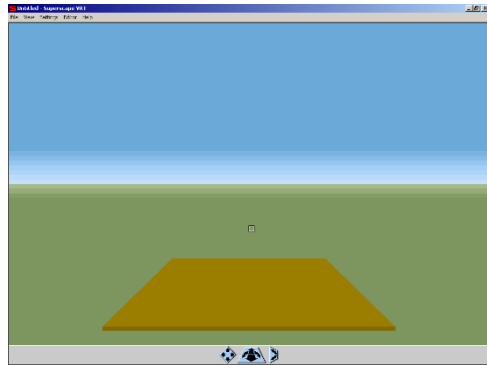


image

Visual Display –A DirectX Rendering Pipeline



Colour Tracking for Camera Control



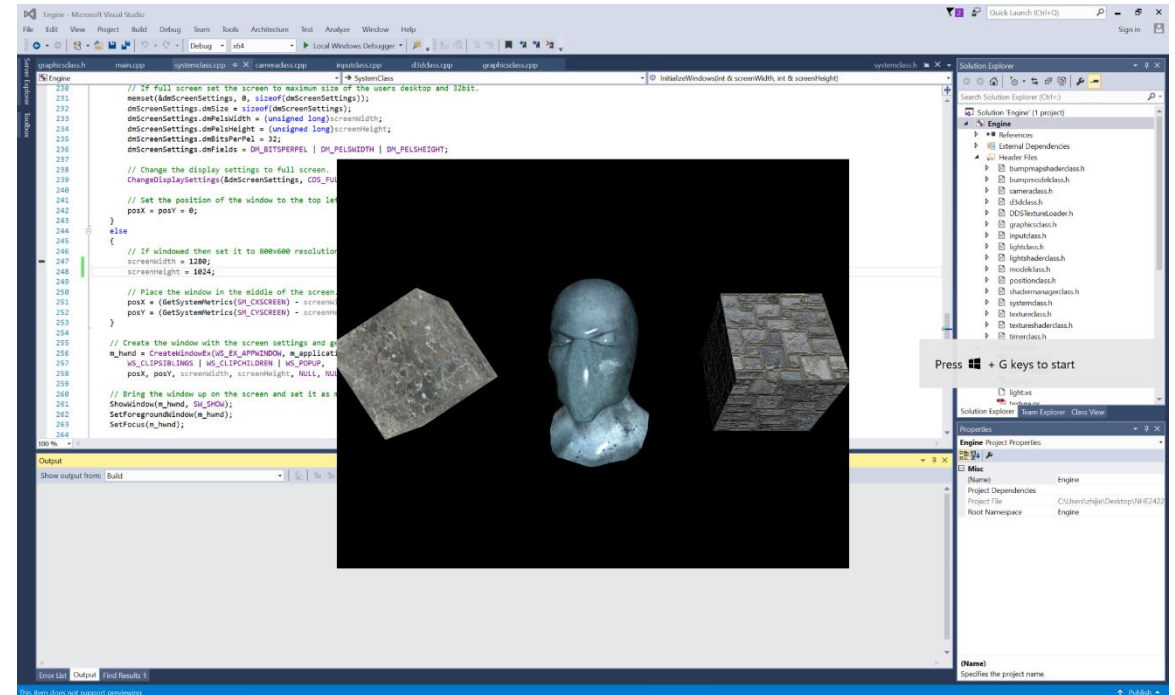
	1	2	3	4
1	Right.x	Up.x	Look.x	0.0f
2	Right.y	Up.y	Look.y	0.0f
3	Right.z	Up.z	Look.z	0.0f
4	-position.right	-position.up	-position.look	1.0f

- For example, to be able to apply our Yaw rotation

- `XMMATRIX yawMatrix;`
- `XMMatrixRotationAxis(&yawMatrix, &up, yaw);`

- To apply yaw, rotate the look & right vectors about the up vector using the yaw matrix:

- `XMVector3TransformCoord\(&look, &look, &yawMatrix\);`
- `XMVector3TransformCoord\(&right, &right, &yawMatrix\);`

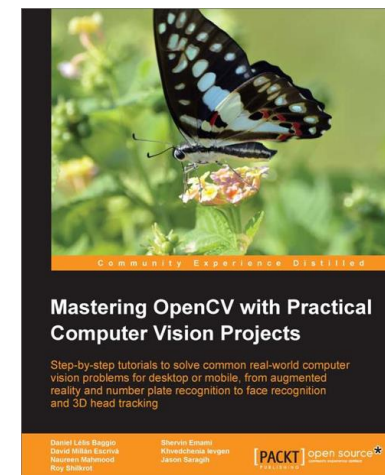
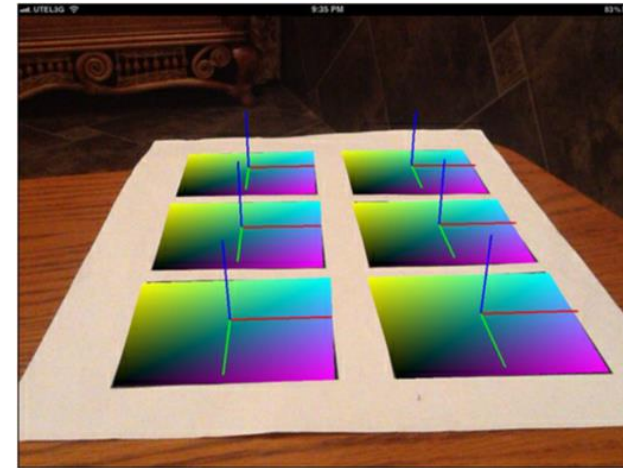


Tutorial 3: Marker-based Augmented Reality

- Marker detection
- Rendering operations

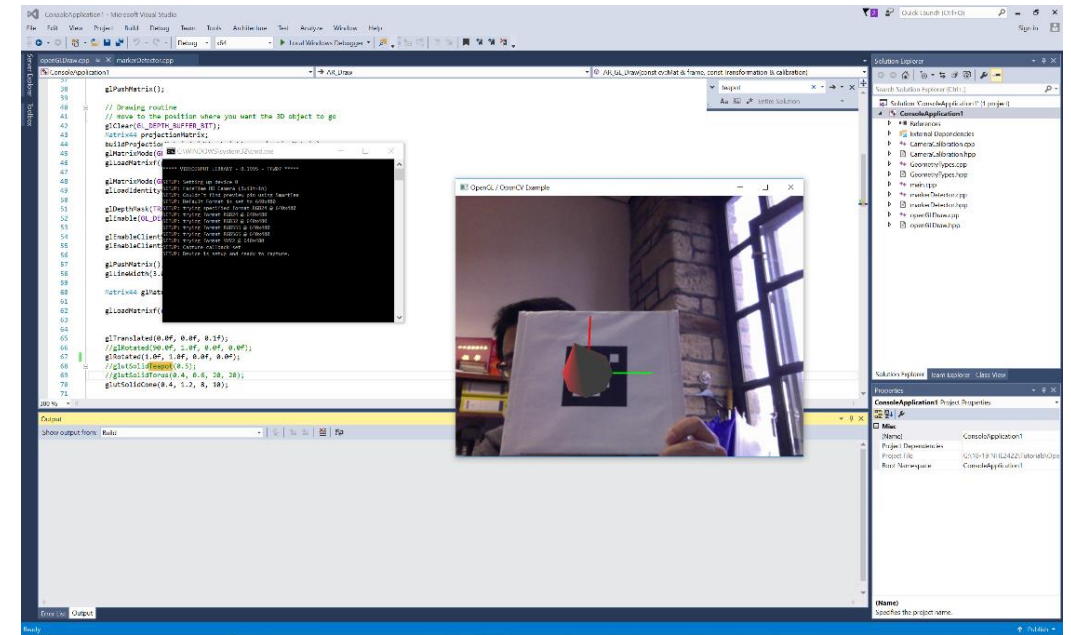
Marker-based AR

- Marker detection:
 - Convert the input image to grayscale;
 - Perform binary threshold operation;
 - Detect contours;
 - Search for possible markers;
 - Detect and decode markers;
 - Estimate marker 3D pose.



Marker-based AR

- Rendering operation:
 - Draw a background image (the last received frame);
 - Copy image data to internal buffers;
 - Processing the new frame and **marker detection**;
 - receive a list of the markers detected on it;
 - pass to the visualization controller (decide what artificial 3D objects to draw);
 - Integration and rendering.



4. What are the potential usages of Desk-top VR?

Discussion session

Open Questions: contents, contents, contents ...

- What fills Virtual Worlds will make or break the VR “frenzy”?
- Can we have genres for VE just like in games?
- Who are the future immersive experience practitioners (designers, developers, users) and how to train them?
- UK Efforts

5. Summary

- VR had been here before ...
- VR is attractive and “cool”
- Not all VR solutions are feasible right now
- But it looks here to stay with new drive from real world applications and off-the-shelf technologies
- **It's a System's perspective this tutorial follows...**