HTML5 + JavaScript = Graphics

Anton Gerdelan
gerdela@scss.tcd.ie

HTML5

- New features for better use on a mobile devices
  - Touchscreen controls
  - Screen rotation handling
  - Automatic full-screen
- <canvas> tag
  - A rectangular area on screen
    <canvas width=512 height=256>
  - Canvas2d and WebGL can plug-in to draw stuff there
- “The web is a platform”

WebGL

- Spin-off of OpenGL 3d vector graphics library
- Previously to do 3d graphics:
  - Compile/create a separate OpenGL build for each platform
    - Apple
    - PC w / different drivers
    - Linux w / different drivers
    - Android
    - iOS
  - Inconistent results on different drivers / platforms

OpenGL

- Started in '90s by SGI as a 3d graphics library
- Key feature – can use accelerated hardware
- GPU -> graphics processing unit
- Graphics rendering is easy to split into many parallel jobs
- GPUs have 100s or 1000s of cores
- = much faster than computing on CPU

WebGL Advantages

- All modern devices have a GPU (including mobiles)
- Capable of very powerful 3d rendering
- Write programme once in JavaScript as a web page
- Runs on everything the same way
  - iOS recently unlocked WebGL
- Use HTML5 as a GUI / interface
- Mix with other web media / web services

Three.js

- A very popular JavaScript helper library for WebGL
- http://threejs.org/
- These demos might work on your phone (depending on network connection in here)
- Twitter integration – consider other web services are easy to tie-in
- Leverage web design for user interfaces

Potential for Business

- Promotions and tangental entertainment
- Advertising
- Information visualisation (Google Maps etc.)
- 3d architectural views – Autodesk 360
  - http://autodesk360.com/

Games

- Concerns around source being visible – hard to commercialise / protect IP
- Last Humble Bundle – WebGL **asm.js**
- Release HTML + js through Steam
- Compiler for C games -> very efficient JavaScript
- Made **millions** in the first days

I Had a Go

- http://antongerdelan.net/mountain_king/
- http://antongerdelan.net/dolphin_rescue/
- Web <audio> tags work – a bit limited so far
- Gamepad interface in next version of HTML (experimental now)
- Multiplayer?

Web Sockets

- Very easy to use from JavaScript
- Asynchronous network interface
- Pass strings to client-server
- Fires a callback function when a message arrives
- “move player 0 to 10,4”
- I wrote a C server – ws handshaking protocol
- Could use socket.js + node.js instead

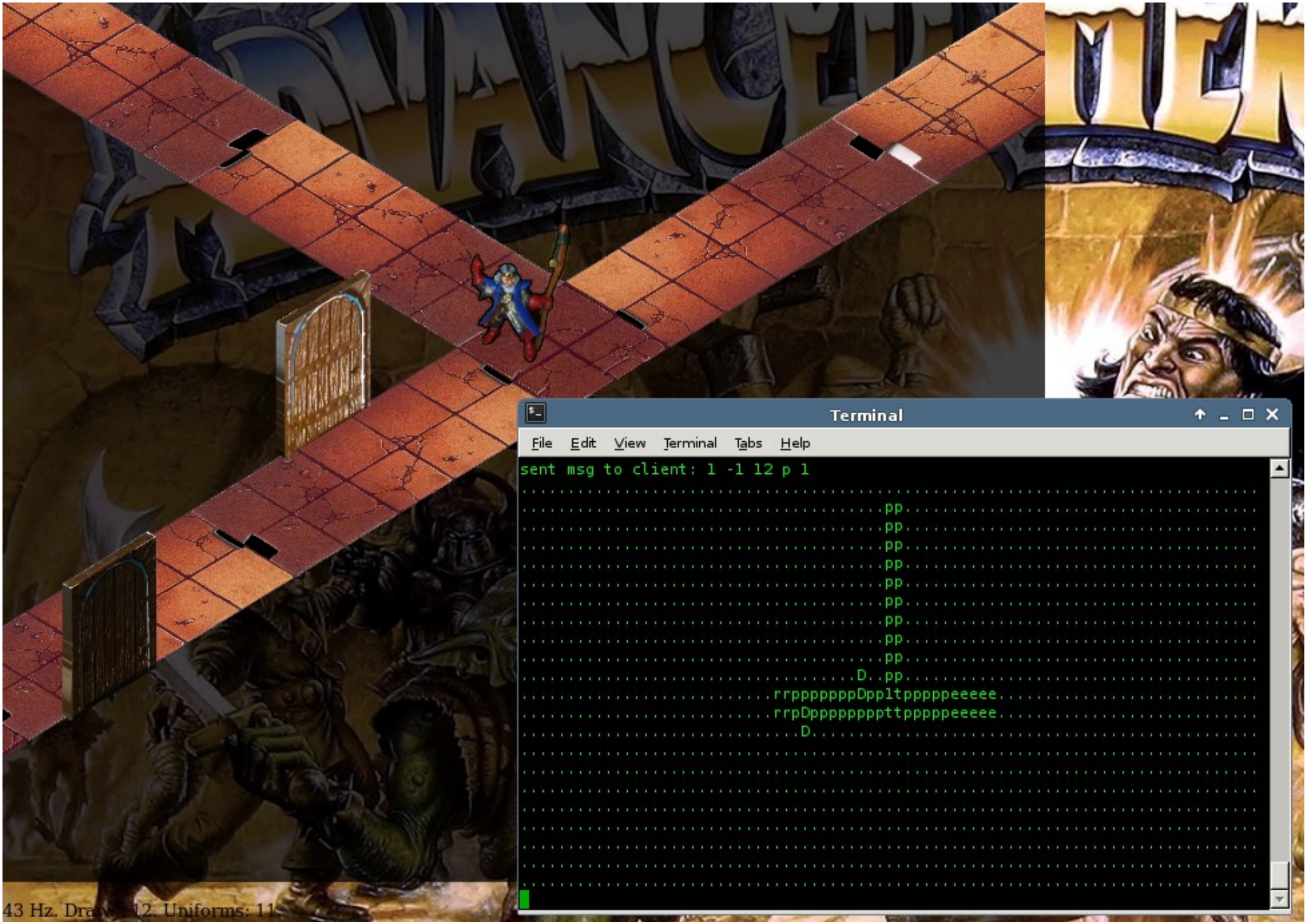

ADVANCED

WS in JS – set up some callbacks

```
54 function ws_connect () {
55 →console.log ("Opening WebSocket connection");
56 →ws = new WebSocket ("ws://localhost:8888");
57 →ws.binaryType = "arraybuffer";
58 →ws.onopen = on_open;
59 →ws.onmessage = on_msg;
60 →ws.onclose = on_close;
61 →ws.onerror = on_error;
62 }
```

WS in JS – incoming message func

```
function on_msg (event) {
	var msg_parts = event.data.split (' ');
	// move
	if (msg_parts[0] == '0') {
		my_player_x = parseInt (msg_parts[1]);
		my_player_y = -parseInt (msg_parts[2]);
		my_player_moved = true;
		//console.log ("rcvd move code, player to " + my_player_x + "," + my_player_y);
	// add tile
	} else if (msg_parts[0] == '1') {
		console.log ("rcvd add tile code");
		//add_tile (x, y, type, rotation)
		add_tile (parseInt (msg_parts[1]), parseInt (msg_parts[2]),
			msg_parts[3], parseInt (msg_parts[4]));
	} else if (msg_parts[0] == '2') {
		console.log ("rcvd add door code");
		//add_door (x, y, rotation)
		add_door (parseFloat (msg_parts[1]), parseFloat (msg_parts[2]),
			parseFloat (msg_parts[3]));
	} else if (msg_parts[0] == '3') {
		console.log ("rcvd change tile type code");
		//change_tile (i, type)
		change_tile_type (parseInt (msg_parts[1]), msg_parts[2]);
	}
}
```


Terminal
File Edit View Terminal Tabs Help
sent msg to client: 1 -1 12 p 1
43 Hz. Draws: 12. Uniforms: 11

Extending to Mobile Devices

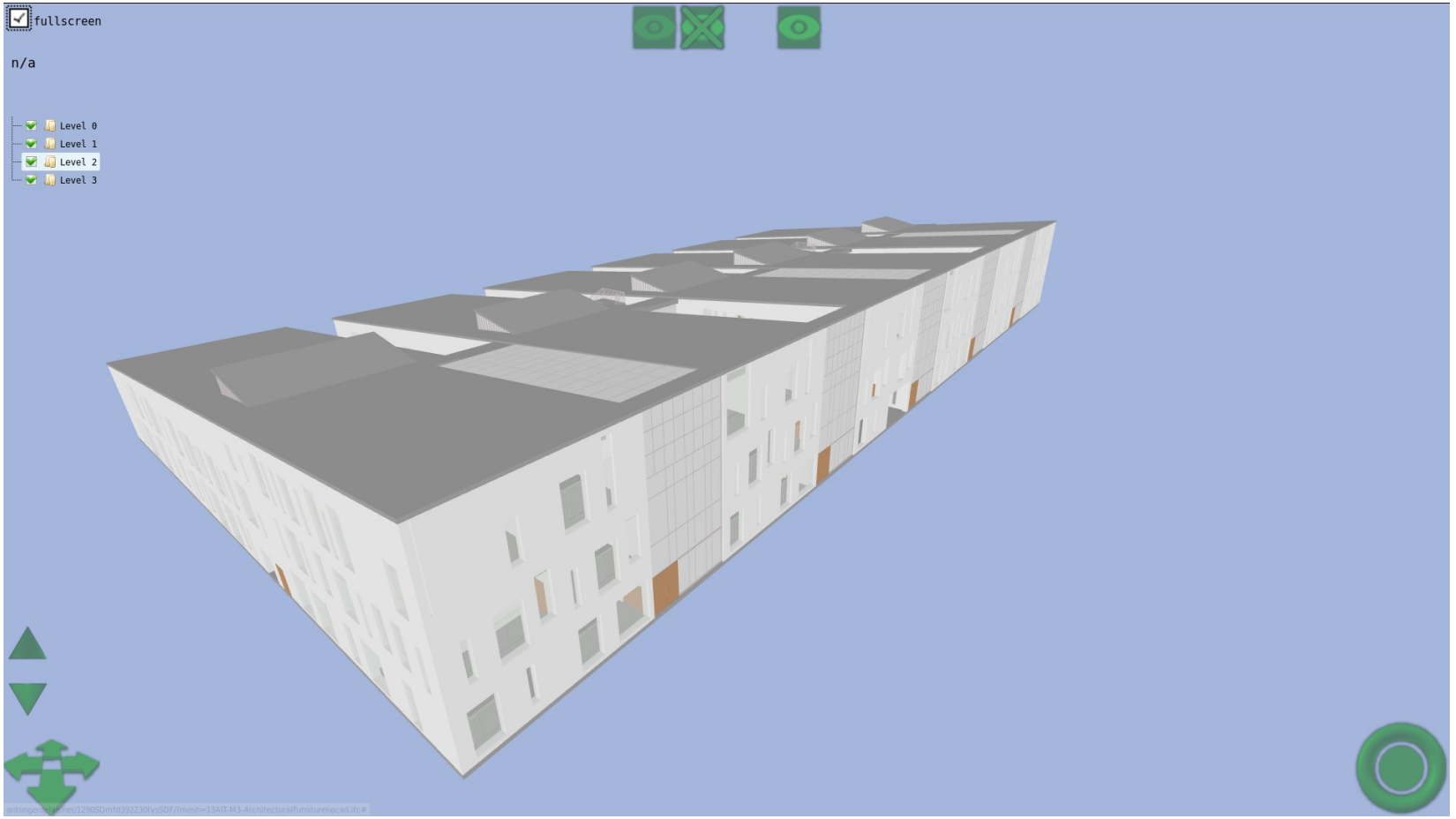
- Can't use keyboard, and mouse emulation isn't always ideal
- Add touch-screen controls
  - Area of screen
  - Make some graphical icons to represent this
  - Have a good idea of ratio pixels : size of thumb

Touch Events

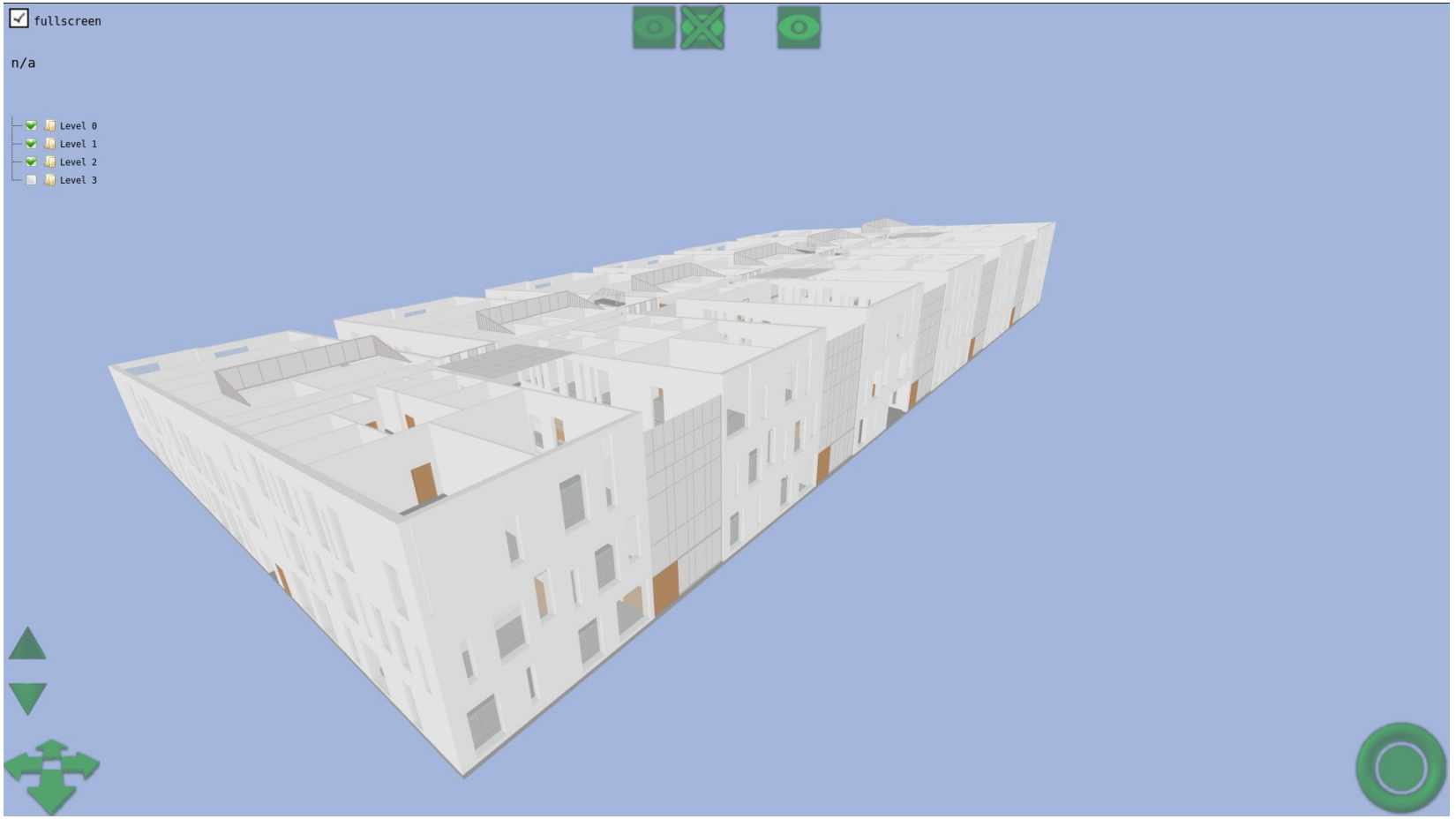
```
function init_touch () {
	//log ("init touch...");
	canvas.addEventListener ("touchstart", touch_start_callback, false);
	canvas.addEventListener ("touchend", touch_end_callback, false);
	canvas.addEventListener ("touchcancel", touch_cancel_callback, false);
	canvas.addEventListener ("touchleave", touch_end_callback, false);
	canvas.addEventListener ("touchmove", touch_move_callback, false);
}
```

- Basically just more callback functions
- Disable the mouse
- These update a variable with the new x,y position
- disable the mouse emulation

Consulting Project

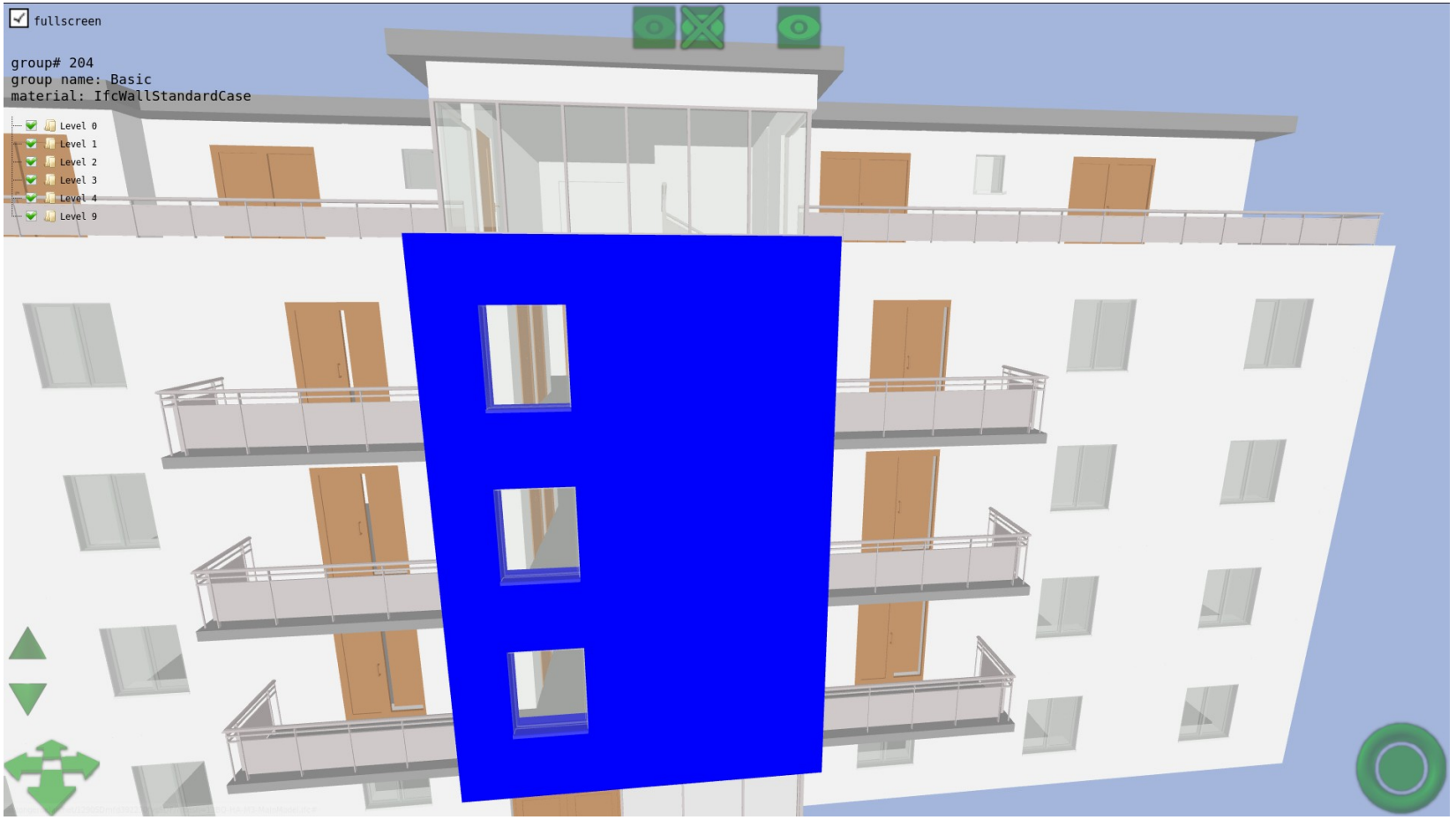


Consulting Project



Consulting Project

Consulting Project



Consulting Project

How WebGL / OpenGL Work

- Vector graphics
  - Define vertex points for each shape
  - Every 3 makes a triangle
- Scales to any size display
- Graphics card has a pipeline to process vertices into 3d shapes and finally 2d shapes
- We write mini-programmes called “shaders”
  - move vertices
  - colour pixel-sized fragments of each triangle

Shaders

- http://shadertoy.com
- Like mini C-programmes
- Parisi - “WebGL Up and Running”
- Cantor and Jones - “WebGL Beginner's Guide”
- Not too hard to learn – and fun!