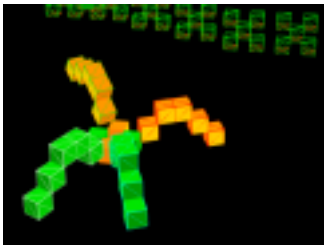
WebGL Spider Model



Arvo Sulakatko (101574IAQDD), Lin Li, Taavi Salumae

jsc-solutions.net

March 15, 2012

Contents

1 The Why **7**
- 1.1 Intro . . . 8
- 1.2 Goals . . . 8

2 The What - Create a WebGL Spider Model **11**
- 2.1 Sensors . . . 12
  - 2.1.1 Light sensors . . . 12
  - 2.1.2 Infrared sensors . . . 12
- 2.2 Programs . . . 12
  - 2.2.1 Program 23 - Calibration . . . 12
  - 2.2.2 Program 43 - Stand . . . 13
  - 2.2.3 Program 53 - Mayday . . . 14
  - 2.2.4 Program 13 - Turn Left . . . 14
  - 2.2.5 Program 14 - Turn Right . . . 14
  - 2.2.6 Program 15 - Go Backwards . . . 14
  - 2.2.7 Program 16 - Go Forwards . . . 14
  - 2.2.8 Program 17 - Go Left . . . 14
  - 2.2.9 Program 18 - Go Right . . . 14
- 2.3 Visual Studio Solution Description . . . 15
  - 2.3.1 Design/Default.htm . . . 15
  - 2.3.2 Library/glMatrix.js . . . 15
  - 2.3.3 Shaders/Geometry.frag . . . 15
  - 2.3.4 Application.cs . . . 15
  - 2.3.5 ApplicationWebService.cs . . . 15

3 The How - Install JSC **17**
- 3.1 Download . . . 17
  - 3.1.1 Agreement and Configuration . . . 18

3.1.2 Start using it 18
3.2 Browser 19

4 Arduino **23**
4.1 Connecting model with a data cable to Arduino 24
4.1.1 Lessons learned - int 25
4.1.2 Lessons learned - function pointers 25
4.1.3 Lessons learned - Serial Port 25

5 References **27**
5.1 Document Source 27
5.2 Project Source 27
5.3 Arduino Sketch Source 27
5.4 Video 27
5.5 JSC Web Installer 28
5.6 Website 28
5.7 Blog 28

List of Figures

1.1 Physical Spider To Be Programmed 7
1.2 What can we see on the spider 8

2.1 Visual Studio Web Developer Express - New Project 11
2.2 Program 18 13
2.3 Solution Explorer 16

3.1 JSC The .NET cross compiler for web platforms 17
3.2 Download JSC at `http://download.jsc-solutions.net` . . 18
3.3 JSC Web Installer 19
3.4 Microsoft Visual Web Developer 2010 Express 20
3.5 Make sure your device is supprting WebGL 21

4.1 Client Server Application Model with USB Serial Connection to Arduino 23
4.2 Control Center - Disconnected 24
4.3 Arduino - spiderwalk2.ino 26

Chapter 1

The Why

[...] people don't buy what you do. people buy why you do it!

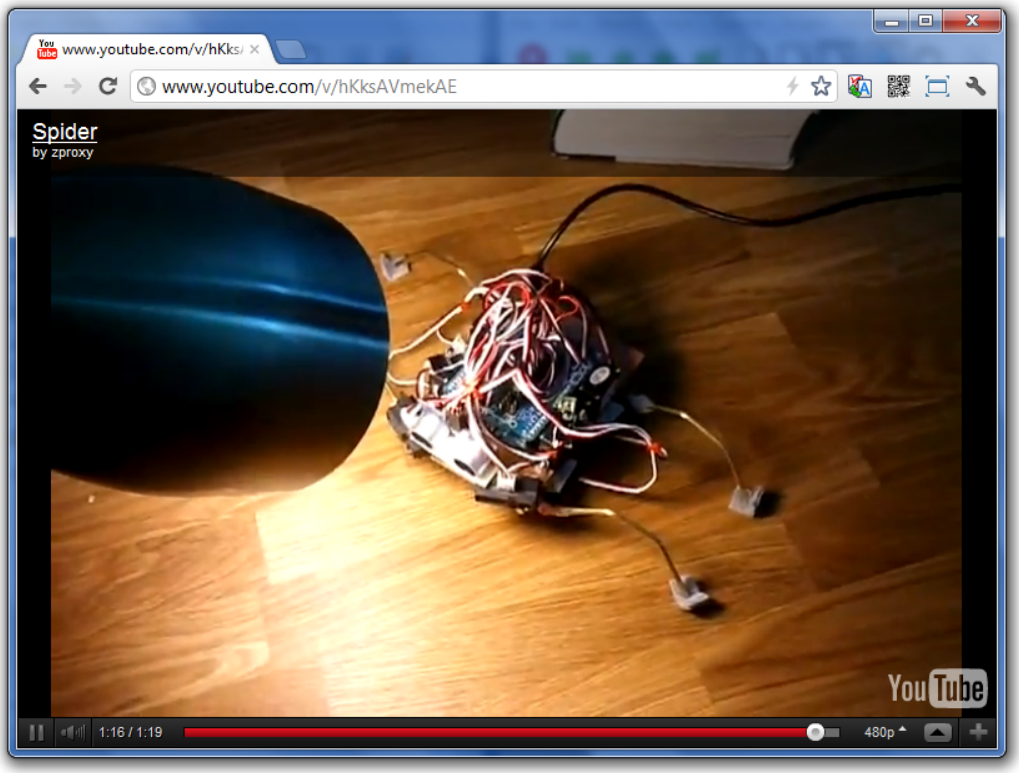


Figure 1.1: Physical Spider To Be Programmed

1.1 Intro

In 2011 I took a course. It was the **Advanced Topics in Biomechanics** course by **Adriano Cavalcanti, Ph.D**. During this course we had to come up with various 3D visualizations of different models. I chose to do that within WebGL. As a final task we had to come up with a mechanic spider. My part was to make it move. I had never programmed a robot before.

I was given a piece of hardware which had a few sensors and four legs.

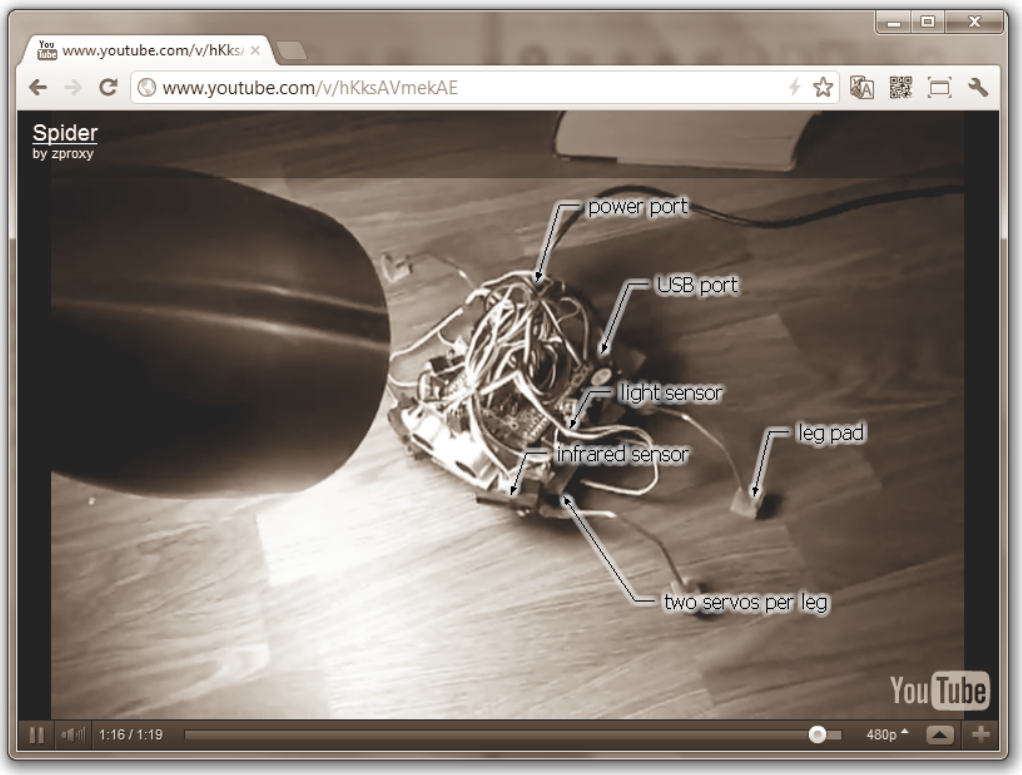


Figure 1.2: What can we see on the spider

1.2 Goals

For a project to be succesful goals needs to be set.

- Avoid obstacles

- Go to the light
- Stop when there

With the current setup we are able to sense light on both sides, sense distance and move servo motors to move the legs. Additionally the spider has to move without a data cable. It can have a power cable attached but cannot have the data cable.

The movement of the legs shall be time dependant. Legs can have different types of movement. We will name them as programs.

Program code was selected at random. Keyboard codes were also considered.

- Program 23 - Calibration
- Program 43 - Stand
- Program 53 - Mayday
- Program 13 - Turn Left
- Program 14 - Turn Right
- Program 15 - Go Backwards
- Program 16 - Go Forwards
- Program 17 - Go Left
- Program 18 - Go Right

While within our model we can visualize any movement we want to we will be limited by the physical version of the spider. For example movements to the left and right will not have the expected outcome.

Next, lets have a look what was built to visualize spider movement.

Chapter 2

The What - Create a WebGL Spider Model

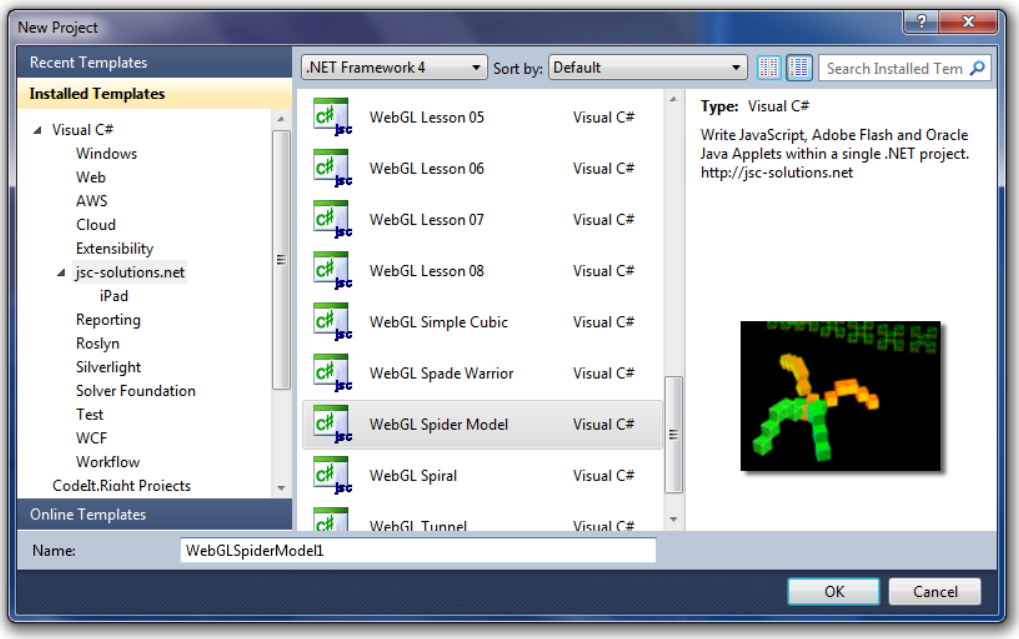


Figure 2.1: Visual Studio Web Developer Express - New Project

While taking this project I wanted to make use of the WebGL technolodgy. During the course itself I had learned how to make a few simple

models. This time I had to do four legs and also add sensor visualization. The project itself is written in CSharp. The JSC compiler will translate it JavaScript to run in a WebGL capable web browser.

Refer to the next section to install jsc!

In this chapter we shall have a look at how to build on this example on your machine.

The project template is part of the JSC experience and as such you will be able to create a new project and go from there.

2.1 Sensors

2.1.1 Light sensors

Within the visualization view we have a white arrow. It will move around based on the two light sensors. Less light detected means the light source is far away. If a strong light signal is to be detected the arrow turns cyan. This would mean spider had reached its destination and should stop. Direction of the light source is inferred by the balance of light signals in the two sensors.

2.1.2 Infrared sensors

Within the visualization view we have a wall in front of the spider. The wall moves near and far based on the signal from both sensors. If for example the lef sensor detected almost near or near obstacle the left part of the wall would move and colorcode itself yellow or red respectivly. Otherwise the wall is green and means there is nothing in the way.

2.2 Programs

2.2.1 Program 23 - Calibration

Calibration mode was used to figure out how much should the legs be able to turn and in what range. Basically I had to tweak the source code of the Arduino Sketch later due to differences with the model.

Figure 2.2: Program 18

2.2.2 Program 43 - Stand

While working with the spider the legs were constantly falling off. They had to be reattached to the servo motors. To know which leg I was referring to in code I had to color code them. I used colors RED, GREEN, BLUE and WHITE. In this program the spider would just stand and try to lift itself up just a bit. It was a known state where I knew in which direction the legs needed to be reattached. Had I not had a known state the legs might have been attached at a wrong angle only to realize the mistake while powering the device itself.

2.2.3 Program 53 - Mayday

Within mayday program the spider sits down and starts wawing all legs at the same time up in the air. It looks like a dance and was used during testing. For example if the spider sensed an obstacle it could just stop and do the mayday dance instead.

2.2.4 Program 13 - Turn Left

One by one all four legs are moved into a new position at left and then the move is finalized by turning all of them into default angle.

2.2.5 Program 14 - Turn Right

One by one all four legs are moved into a new position at right and then the move is finalized by turning all of them into default angle.

2.2.6 Program 15 - Go Backwards

While the spider cannot sense anything at his back it can walk in reverse. This was used while testing.

2.2.7 Program 16 - Go Forwards

The most popular yet not that interesting movement is just to go forwards.

2.2.8 Program 17 - Go Left

While the model can visualize movement to the left it was not used as the physical model did not go left.

2.2.9 Program 18 - Go Right

While the model can visualize movement to the right it was not used as the physical model did not go right.

2.3 Visual Studio Solution Description

2.3.1 Design/Default.htm

Each client server web application begins with a default HTML page with the initial design. If the bowser supports JavaScript the client side code will be loaded and it has the freedom to dynamically change the document. In our case we created a 3D canvas.

2.3.2 Library/glMatrix.js

While JSC has the concept of ScriptCoreLib where all frequently used code is hosted it does not yet have 3D math support. The glMatrix library allows us to do our Vector and Matrix calculations.

2.3.3 Shaders/Geometry.frag

Until now code in the client ran on a CPU. With WebGL we can run code on a GPU. The fragment and vertex shaders are somewhat simple programs to be run on the client GPU.

2.3.4 Application.cs

This is the client side code compiled into JavaScript. It is also responsible for the 3D visualization.

2.3.5 ApplicationWebService.cs

This is the server side code. We are not actually using it in this example.

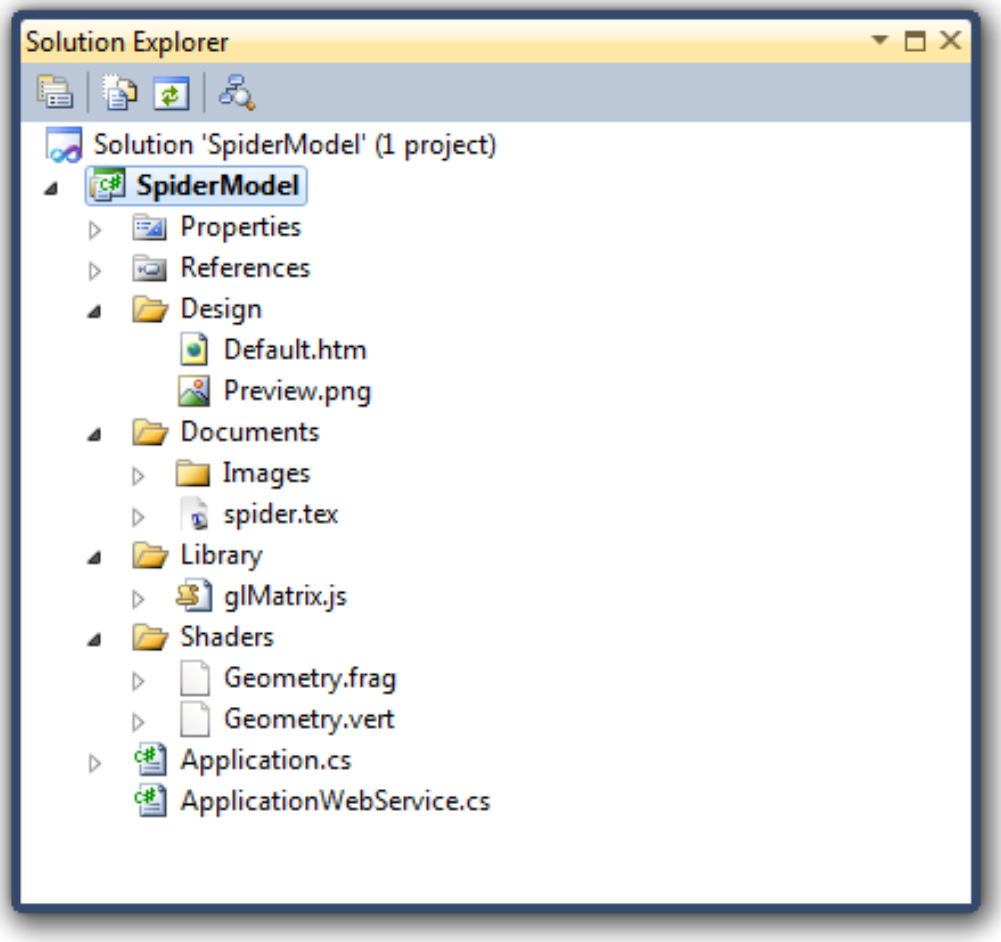


Figure 2.3: Solution Explorer

Chapter 3

The How - Install JSC

JSC will allow to program WebGL applications in CSharp

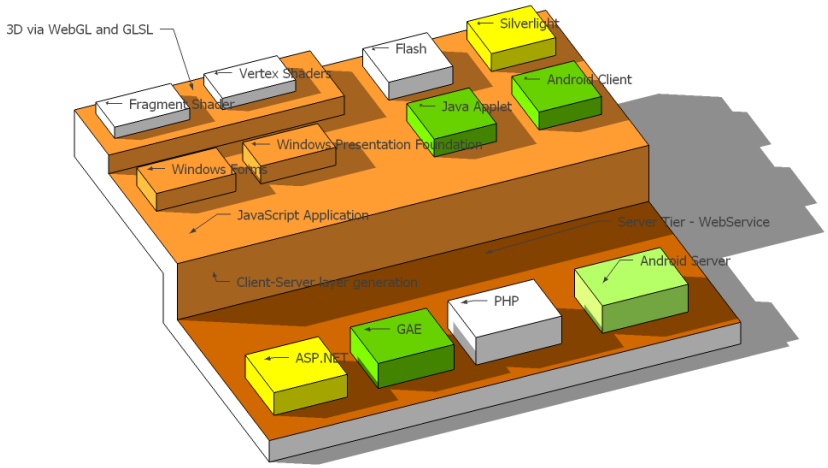


Figure 3.1: JSC The .NET cross compiler for web platforms

Installing JSC is easy. Before you do make sure you have installed Microsoft Visual Web Developer 2010 Express.

3.1 Download

At our webpage we have a link to JSC Web Installer. It is a ClickOnce application and supports features like prerequiste installation and updates.

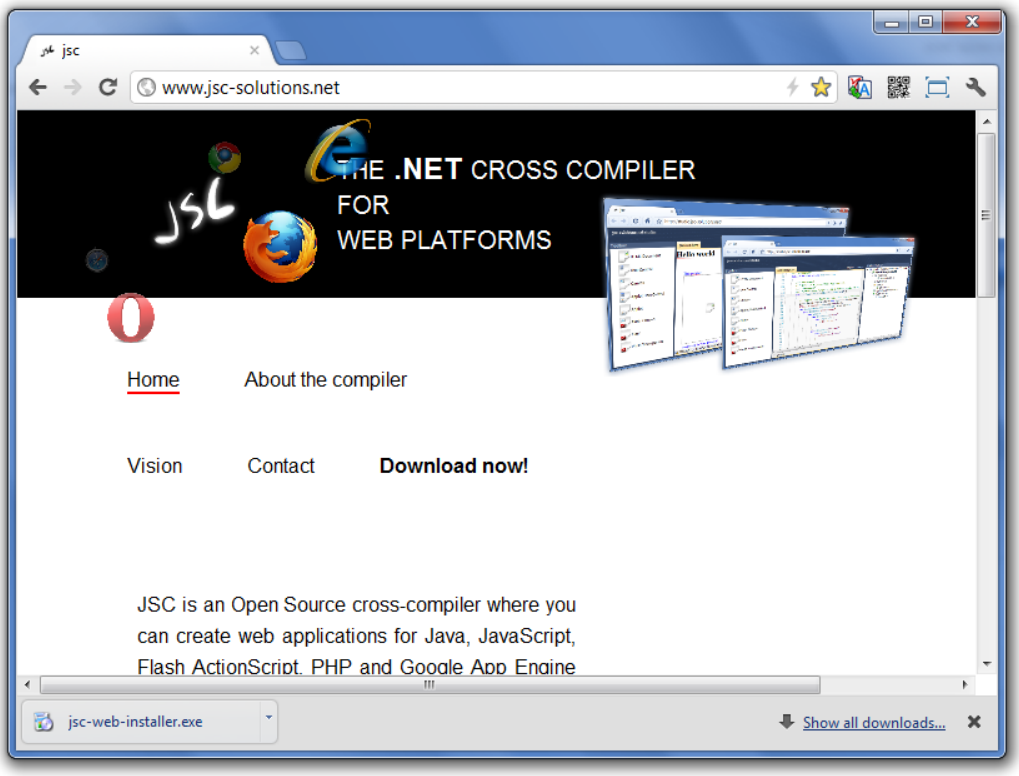


Figure 3.2: Download JSC at `http://download.jsc-solutions.net`

3.1.1 Agreement and Configuration

At this time JSC is distributed for evaluation purposes. You will be asked to read and agree the license. Later you will find new project templates for Visual Studio. WebGL Spider Model being one of them. After installation some configuration options will become available. JSC integrates with third party compilers and SDKs like Adobe Flash and Oracle Java. Unfortunatly this will be out of scope of this document.

3.1.2 Start using it

Refer back to the previous chapter to work with a JSC Web Application project for more details.

Figure 3.3: JSC Web Installer

3.2 Browser

For older machines WebGL might need additional manual configuration.

```
chrome.exe -enable-webgl -enable-apps -ignore-gpu-blacklist
```

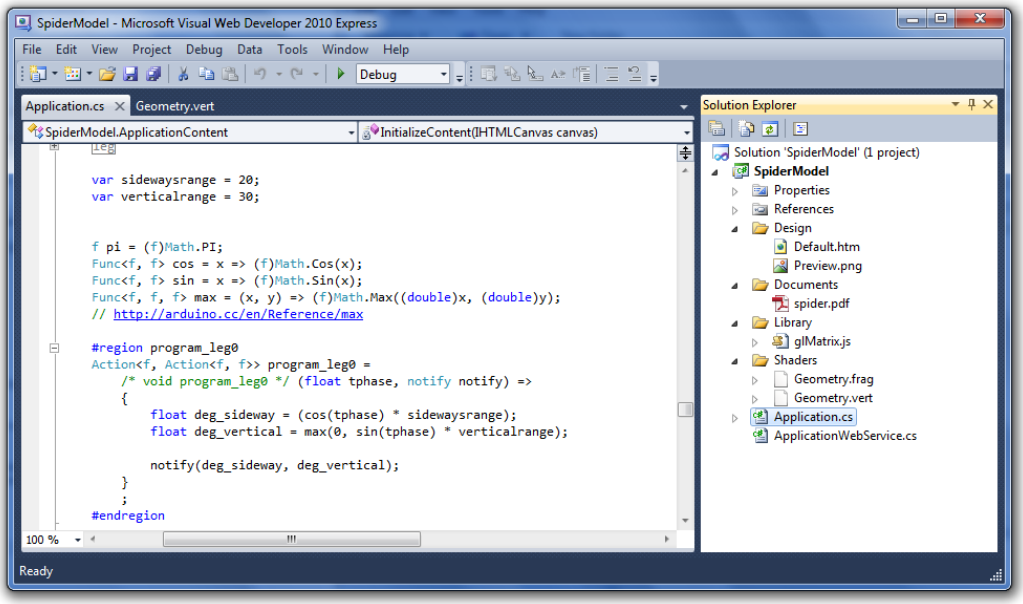


Figure 3.4: Microsoft Visual Web Developer 2010 Express

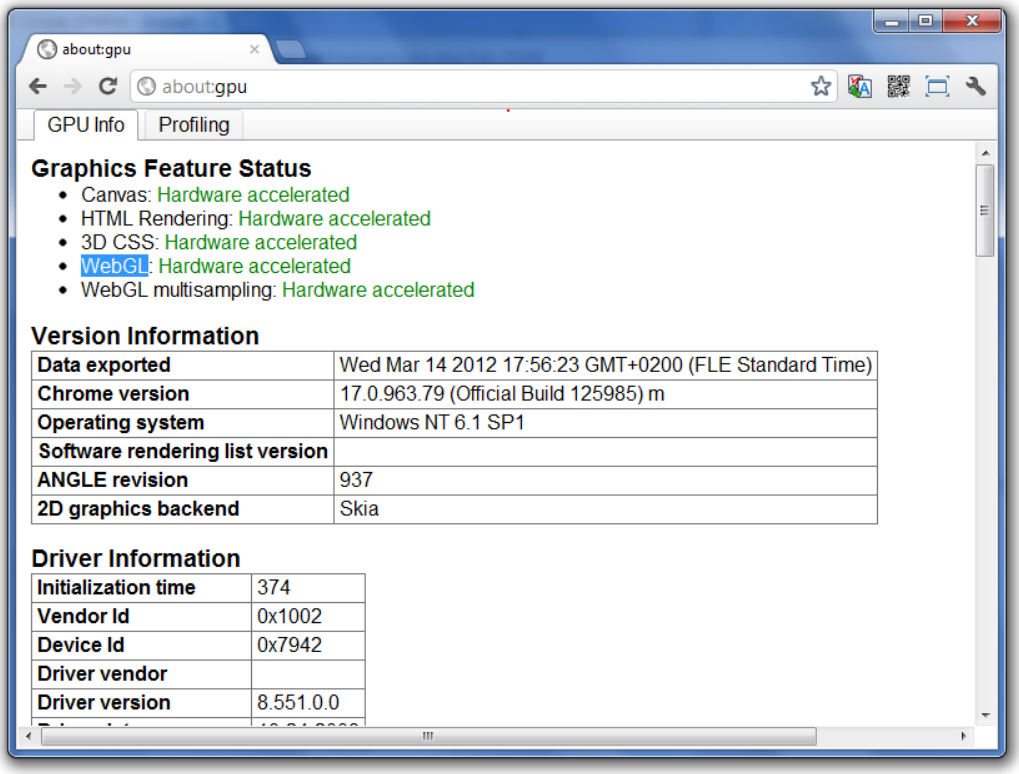


Figure 3.5: Make sure your device is supprting WebGL

Chapter 4

Arduino

Although this document briefly describes Arduino related development it is considered out of scope and is not part of the default **jsc eXperience**.

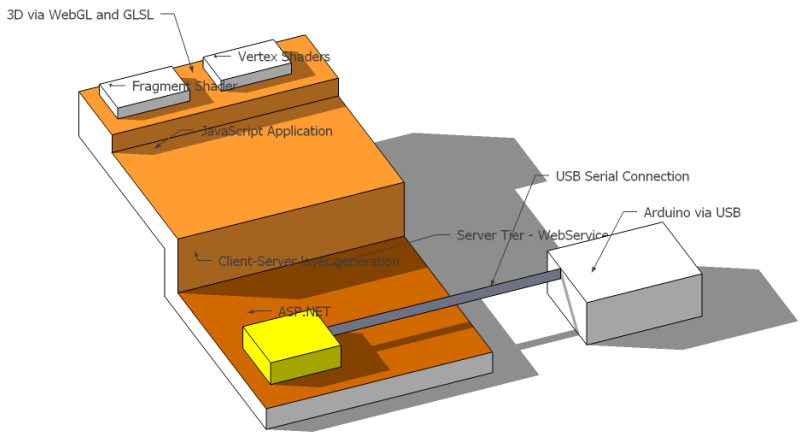


Figure 4.1: Client Server Application Model with USB Serial Connection to Arduino

4.1 Connecting model with a data cable to Arduino

This model was extended in a related project to connect to the Arduino via USB Serial Port. The spider is listening for Program Override code. This allows to issue specific commands to the spider and test out new ideas.

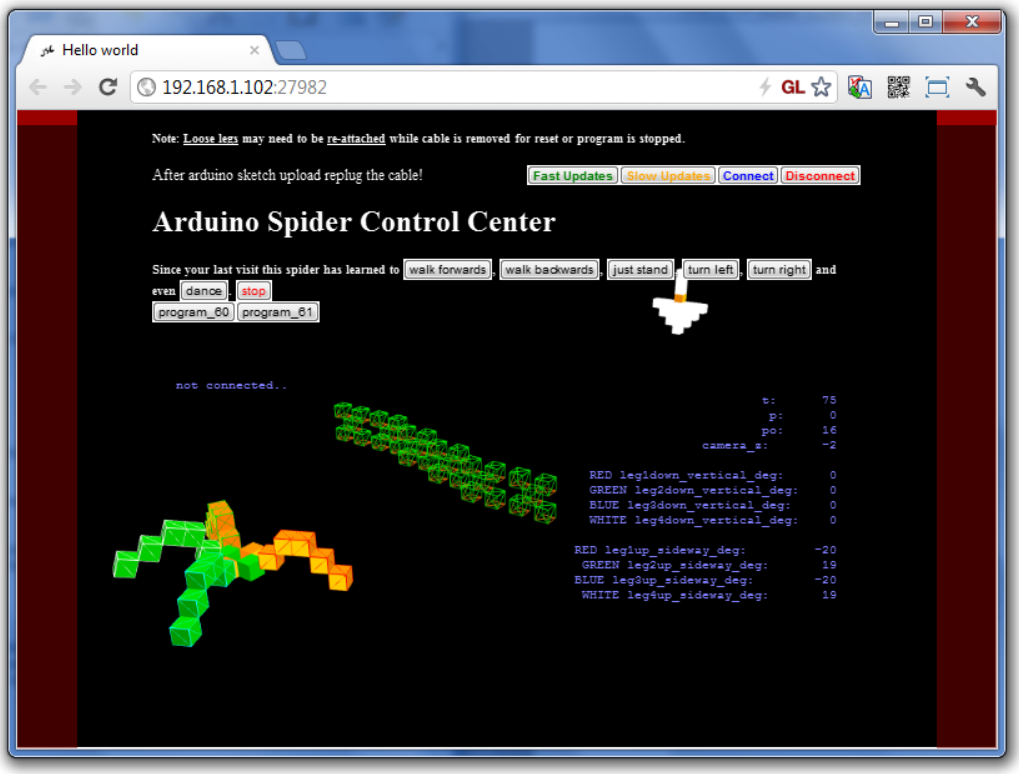


Figure 4.2: Control Center - Disconnected

At this time jsc does not support any languages that target Arduino platform. As such I had to make use of Arduino programming language. Otherwise I could of had my CSharp code compiled to Arduino. This would of had allowed me to use the same code in the visualization and on the chip.

4.1.1 Lessons learned - int

While programming for Arduino I had to manually port my code I had written for the visualizer to the Arduino platform. In doing so I discovered that the int is considered to be 16 bits. To overcome that I had to divide before I did my multiplication. Yes I had to track down an overflow bug before I realized this.

4.1.2 Lessons learned - function pointers

At some point I realized that I cannot make use of function pointers with current hardware version. The callbacks I used were simple. They only had a few parameters. This allowed me to replace the function pointer with pointer to variable and have the same behaviour of code.

4.1.3 Lessons learned - Serial Port

When a new Arduino Sketch was uploaded to the device the USB cable had to be reseated. Otherwise the connection was showing garbage data. Printing to the serial is expensive. While debugging the spider I was printing out so many bytes that a smooth walking animation almost came to a halt. The workaround is to print as little data as possible to the serial.

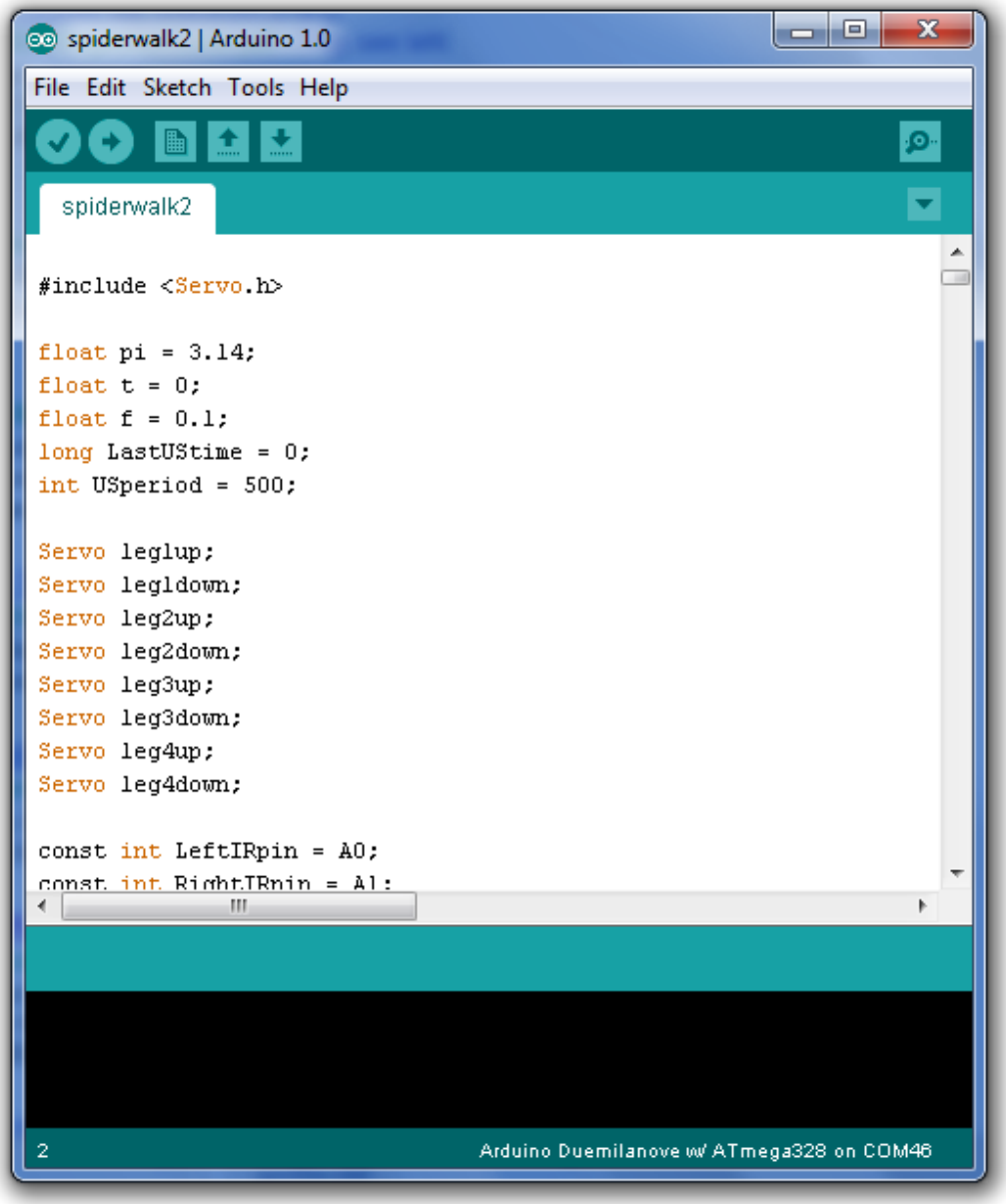


Figure 4.3: Arduino - spiderwalk2.ino

Chapter 5

References

5.1 Document Source

https://jsc.svn.sourceforge.net/svnroot/jsc/examples/javascript/ArduinoSpiderControlCenter/SpiderModel/Documents/spider.tex

5.2 Project Source

https://jsc.svn.sourceforge.net/svnroot/jsc/examples/javascript/ArduinoSpiderControlCenter/SpiderModel/

5.3 Arduino Sketch Source

https://jsc.svn.sourceforge.net/svnroot/jsc/examples/javascript/ArduinoSpiderControlCenter/ArduinoSpiderControlCenter/ArduinoSketches/spiderwalk2/spiderwalk2.ino

5.4 Video

http://www.youtube.com/v/hKksAVmekAE

5.5 JSC Web Installer

http://download.jsc-solutions.net

5.6 Website

http://www.jsc-solutions.net

5.7 Blog

http://zproxy.wordpress.com