# 600086 Lab Book

# Week 4 – CUDA Lab 4. CUDA OpenGL Interoperability & Image processing

Date: 24th Feb 2022

## Exercise 1. Create an OpenGL-CUDA program based on a CUDA SDK sample

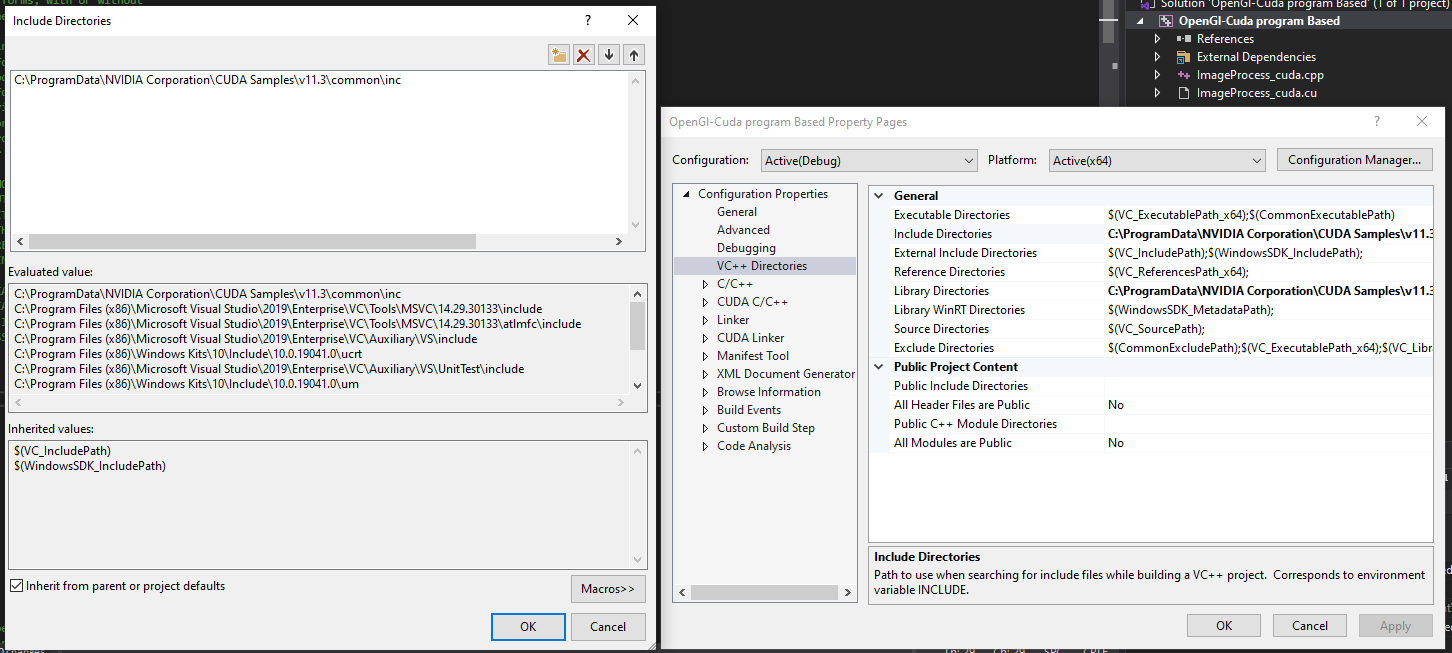
### Question:

Create an OpenGL-CUDA program based on a CUDA SDK sample

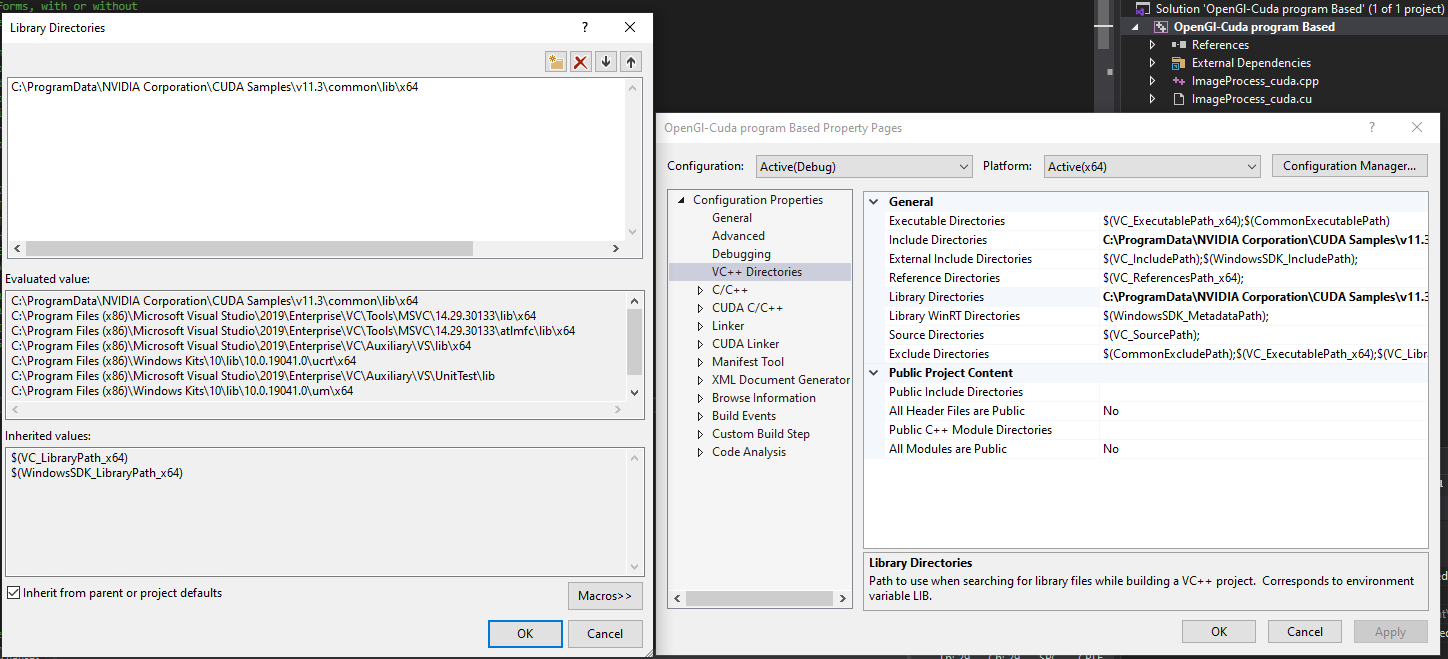
### Solution:

No sample code to show

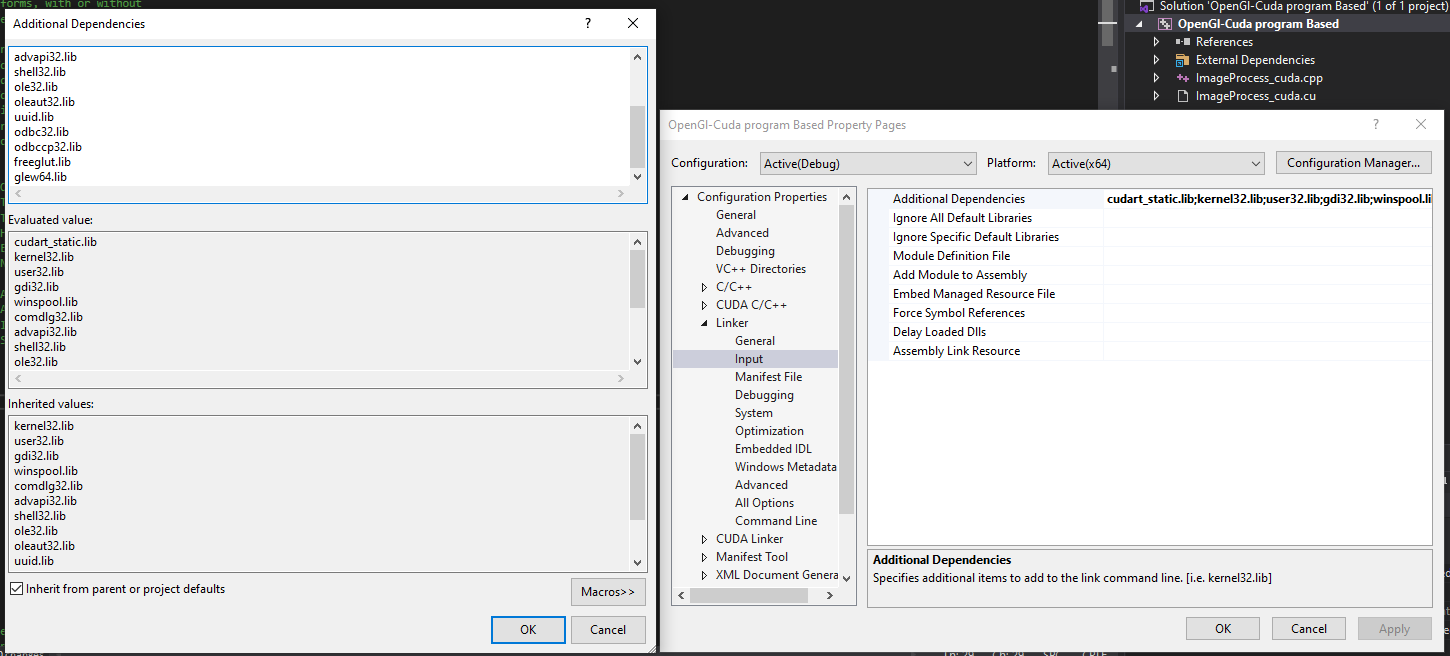
Step 1 : adding include directories to project



Step 2 : adding Lib directories



Step 3 : adding to the Linker files



Step 4 : I then compiled the project resulting in the command line output shown in sample output data

### Test data:

n/a

### Sample output:

### 

### Reflection:

Nothing to report was fairly perfunctory

### Metadata:

N/A

### Further information:

N/A

## Exercise 2. Understand pixel colour

## Question:

a) An image is simply a 2D array of pixels. Each pixel has a colour value which can be digitally

represented as a list of numbers, depending on the data format adopted. In the framework, the

Colour of each pixel is represented in RGBA format using 4 integers, each of which ranging from 0

to 255. Open ImageProcess\_cuda.cu and go to the method d\_render( ), modify the 4 numbers

shown in make\_uchar4( ..., ..., ..., ... ) in the following line:

d\_output[i] = make\_uchar4(c \* 0xff, c \* 0xff, c \* 0xff, 0);

say,

d\_output[i] = make\_uchar4(0xff, 0, 0, 0);   
 and then

d\_output[i] = make\_uchar4(0, 0xff, 0, 0);   
d\_output[i] = make\_uchar4(0, 0, 0xff, 0);

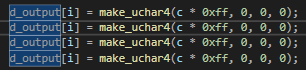
b) The original image is a grey value image, the pixel intensity at a pixel position at (u,v) is read using float c = tex2DFastBicubic(texObj, u, v); where c is in [0, 1].

c) Now modify the value d\_output[i] using image pixel value c read from image location at

(u, v) with the following colour and observe how the image colour is changed.

d\_output[i] = make\_uchar4(0, 0, c\*0xff, 0);

### Solution:



Running each of these one at a time and commenting out the other to display the different resulting outcomes

### 

### Test data:

n/a

### Sample output:

|  |  |
| --- | --- |
| code | output |
| d\_output[i] = make\_uchar4(0xff, 0, 0, 0); |  |
| d\_output[i] = make\_uchar4(0, 0xff, 0, 0); |  |
| d\_output[i] = make\_uchar4(0, 0, 0xff, 0); |  |
| d\_output[i] = make\_uchar4(0, 0, 0xff, 0); |  |

### Reflection:

Nothing to report was fairly perfunctory

## Exercise 3. Image Transformation

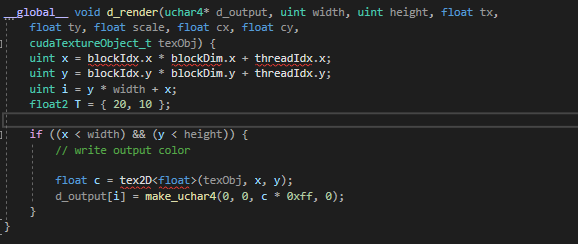
## Question:

Demonstrate Image transformation.

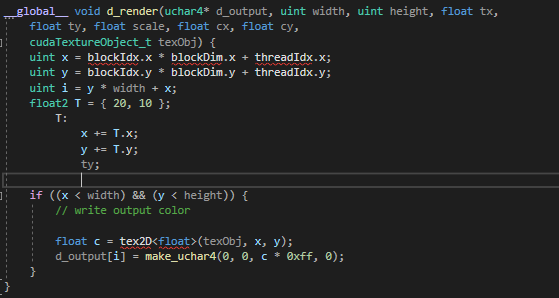
### Solution:

Translate the image.

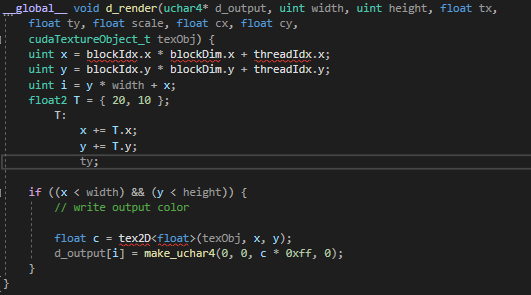
1. Define a translation as a 2D vector, say float2 T={20, 10};



1. Translate (x, y) with vector T: x +=T.x; y +=T.y;



1. Read pixel colour with translated coordinates x, y: float c = tex2D(texObj, x, y);



1. Compile the run your program and observe if the image is translated according to your wish.
2. Observe how the image is transformed by defining different translation vectors.

### Sample output:

|  |  |
| --- | --- |
| Translation | result |
| float2 T = { 20, 10 }; |  |
| float2 T = { 200, -100 }; |  |

### Reflection:

The image has been translated by moving the image in the way described in the vector T. the first value adjusts the translation in the xs axis and the second value adjusts it in the Y axis. When translating to the right the image is replaced by black plixels where the image has moved but when translating to the left the image appears stretched.

### Further information:

Why doe the image appear stretched whgen translating in negative directions?

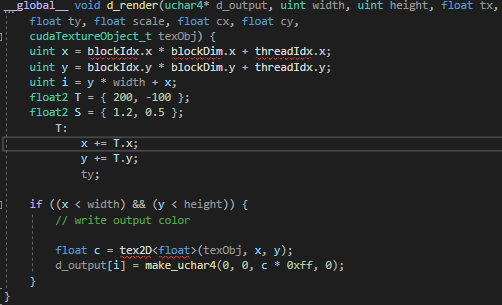
## Question:

Demonstrate Image Scaling

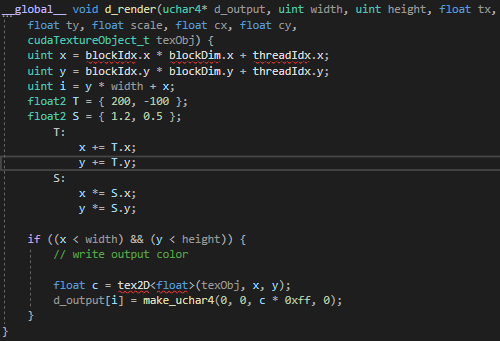
### Solution:

Scale the image

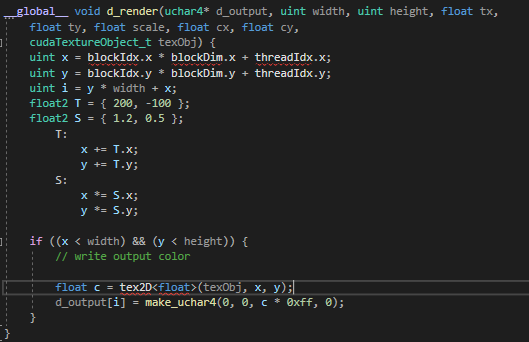
1. Define a scaling transformation as a 2D vector, say float2 S= {1.2, 0.5};



1. Scale (x, y) with vector S: x \*=S.x; y \*=S.y;



1. Read pixel colour with scaled coordinates x, y: float c = tex2D(texObj, x, y);



1. Compile the run your program and observe if the image is scaled according to your wish.
2. Observe how the image is scaled by defining different scaling vectors.

### Sample output:

|  |  |
| --- | --- |
| Translation | result |
| float2 S = { 1.2, 0.5 }; |  |
| float2 S = { 2, -0.5}; |  |

### Reflection:

The image has been translated by scalin it according to the vector S in the second image I experimented by using a negative value this resulted in a strange image

### Further information:

Why does the image appear as shown when scaled in negative directions?

## Question:

Demonstrate Image Rotation

### Solution:

Rotate the image

1. Define a rotation matrix for a certain rotation angle, float angle = 0.5;

Text

Description automatically generated

1. Rotate (x, y) with rotation matrix defined below:

Text

Description automatically generated

1. Read pixel colour with scaled coordinates

Text

Description automatically generated

1. Compile the run your program and observe if the image is rotated according to your wish.
2. Further observe how the image is rotated by defining different rotation angles.

### Sample output:

|  |  |
| --- | --- |
| Translation | result |
| float angle = 0.5; |  |
| float angle = 45; |  |
| float angle = 1; |  |
| float angle = -0.5; |  |

### Reflection:

The image has been around the origin which is the top left however I am unsure what the angel value equates to it cant be an angle in degrees as 45 results in the same result as 1.

### Further information:

What unit does the angle float represent?

## Question:

Demonstrate scaling by position

### Solution:

Implement the following struct:

Text

Description automatically generated

Now the image can be edited by adjusting the values passed into the kernel as shown below:

Text

Description automatically generated

### Sample output:

|  |  |
| --- | --- |
| Translation | result |
| scale = 1.3,  cx = 35,  cy = 0; |  |
| scale = 0.5,  cx = 35,  cy = 1; |  |

### Reflection:

Fairly simple

### Further information:

None.

## Question:

Demonstrate rotation about a centre point

### Solution:

Modify the rotation code to incorporate the passed in centre point values.

Text

Description automatically generated

### Sample output:

|  |  |
| --- | --- |
| Translation | result |
| angle = 0.5,  cx = width/2,  cy = height/2; |  |
| angle = 0.5,  cx = -200,  cy = 150; |  |

### Reflection:

The rotation calculation had to be adjusted to account for the centre point this is done by performing the calculation on the x value – the cx value and the y – cy value and then the original c values added back to the result as shown in the code sample.

### Further information:

None.

## Question:

Demonstrate simplified translation implementation.

### Solution:

The below code implementation will translate the image based on the passed in bvariable to the kernel

Text

Description automatically generated

### Sample output:

|  |  |
| --- | --- |
| Translation | result |
| scale = 1,  angle = 0.5,  cx = width/2,  cy = height/2,  tx = 100,  ty = -100, |  |
| scale = 1,  angle = 0.5,  cx = width/2,  cy = height/2,  tx = -50,  ty = 50, |  |

### Reflection:

Fairly perfunctory

### Further information:

None.

## Exercise 4. Image smoothing

## Question:

Demonstrate Image smoothing.

### Solution:

The below code implements image smoothing using an order 1 square neighbour for reference

Text

Description automatically generated

### Sample output:

|  |  |
| --- | --- |
| Translation | result |
| scale = 1,  angle = 0.5,  cx = width/2,  cy = height/2,  tx = 100,  ty = -100, |  |

### Reflection:

Fairly perfunctory in order to adapt this to smooth by any order then a 2d array of pixels could be used instead of the concrete 5 variable implementation used here. The size of the order of neighbours would need to be passed in to the kernel as an additional variable

### Further information:

None.