

Robotics & Automation

ENGT4314

Tutorial 4

Homogeneous Coordinates - 2D Transformations

Translation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & d_x \\ 0 & 1 & d_y \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Scaling

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} s_x & 0 & 0 \\ 0 & s_y & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

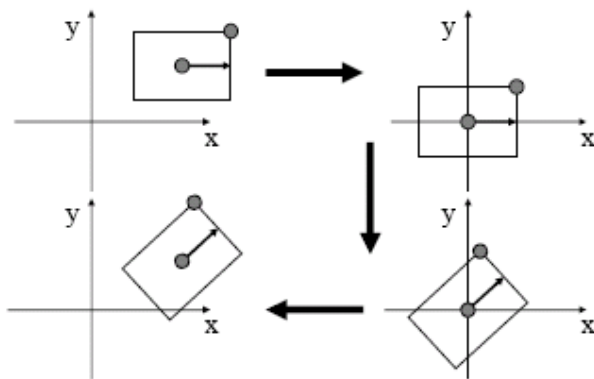
Rotation

$$\begin{bmatrix} x' \\ y' \\ 1 \end{bmatrix} = \begin{bmatrix} \cos \theta & -\sin \theta & 0 \\ \sin \theta & \cos \theta & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}$$

Matrix Composition

Transformations can be combined by matrix multiplication

Warning!: *order of transformations matters*



Example: Rotating about arbitrary point; Translate back to origin, rotate, translate back to original position.

$$\mathbf{p}' = (\mathbf{TRT}) * \mathbf{p}$$

Q: A triangle has vertices at (0,0), (0,2) and (1,0).

1. Draw the triangle
2. Write the transformation matrix to translate the triangle 1 unit to the right. Perform the matrix multiplications, and draw the result.
3. Write the transformation matrix to scale the triangle by 2 on the x-axis, and 1.5 on the y axis. Perform the matrix multiplications, and draw the result.
4. Write the transformation matrix to rotate the triangle by 90 degrees, anti-clockwise. Perform the matrix multiplications, and draw the result.
5. Combine parts 2 & 3, to translate the object and then scale it. Calculate the combined matrix transformation. Perform the matrix multiplications, and draw the result.

Homogeneous Coordinates - 3D Transformations

Translation	Scaling	(note Z-rotation similarity to 2d)
$\begin{bmatrix} 1 & 0 & 0 & d_x \\ 0 & 1 & 0 & d_y \\ 0 & 0 & 1 & d_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} s_x & 0 & 0 & 0 \\ 0 & s_y & 0 & 0 \\ 0 & 0 & s_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	
Rotation (x)	Rotation (y)	Rotation (z)
$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos \theta & -\sin \theta & 0 \\ 0 & \sin \theta & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos \theta & 0 & \sin \theta & 0 \\ 0 & 1 & 0 & 0 \\ -\sin \theta & 0 & \cos \theta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$	$\begin{bmatrix} \cos \theta & -\sin \theta & 0 & 0 \\ \sin \theta & \cos \theta & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$

Q: A triangle has vertices at (0,0,0), (0,2,0) and (1,0,-1).

1. Draw the triangle, and an XYZ axis
2. Write the transformation matrix to translate the triangle 1 unit to the right. Perform the matrix multiplications, and draw the result.
3. Rotate the triangle by 90 degrees around the X axis
4. Rotate the triangle by 90 degrees around the Y axis

Arrays:

I have noticed that some students have had issues with strings and arrays, so I present a small crash course here.

Arrays are simply consecutively stored elements, to declare an array we use the following syntax.

```
<type> <variable>[<number of elements>];
```

For example

```
int foo[5]; // An array that contains 5 integers.
```

The first element always starts at 0, foo is simply a pointer to the first element in the array. To access elements we dereference using [.] , for example.

```
foo[2] = 4; // sets the 3 element (0,1,2,...) to 4.
```

The above operation is equivalent to $*(foo + 2) = 4$; Refer to tutorial 1 for more information on pointers.

Arrays are particularly useful inside loops for example

```
for (i = 0; i < 5; i++) {  
    printf("Element %d = %d\n", i, foo[i]);  
}
```

Which would print each element on its own line.

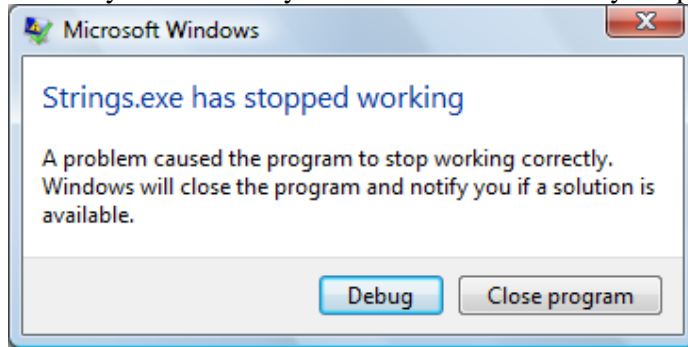
The c++ compiler by default does not initialize the values in an array and you should assume that they are random; you can however initialize an array by the following syntax.

```
foo[5] = {9,8,3,2,6};
```

The number of elements is optional and the compiler will use the number of elements inside {...} if omitted.

A word of warning about using arrays: C++ does not check if you try to access an element outside of you array eg. In the above example foo[9]. If you do access an element outside of the array you will be accessing the memory that exists after the array, now one of two things will happen. If the memory accessed outside the array belongs to the same app and is not restricted you will not receive an error and will be working with that particular value. This can obviously cause bugs in parts of your code that seem unrelated to the code that was altering the memory and thus can be difficult and time consuming to debug, out of bounds memory locations can also be of any type and thus may cause additional strange behavior.

Secondly if the memory is in a restricted location you application will generate a seg-fault.



Or in more descriptive versions of windows you may receive a message similar to “An application.exe tried to access memory location 0x..... the location could not be read/write”. The same will occur if you use a pointer with an invalid address, as some of you discovered in lab2 when eyeSim would crash if fopen failed and returned a NULL pointer.

Strings

A string is simply an array of characters, except that the last character must always be a null character (`\0`). Null characters terminate strings and allow function to determine where the end of the string is, (Remember that the bounds of an array are not usually available natively). We can define strings like following:

```
Char str1[] = "Hello";
```

Hence `str1` will be an array of length 6 elements of type `char`, the array contains `{ 'H', 'e', 'l', 'l', 'o', '\0' }`. It is important to remember to make your strings one element larger than the number of characters so that the NULL character can be stored. If there is no NULL character stored in the array string function may continue to read until it finds one and thus will attempt to access memory outside of the array.

Question:

Will `if (str1 == str2) {}` work as expected (checking if two strings are equal)? If not what does this `if` statement achieve?

String Functions

The <string.h> library contains a number of functions that operate on strings. Below are a few of them.

char *strcat(char *dest, const char *src); appends the string src to dest

int strcmp(const char *, const char *); compares two strings numerically

char *strcpy(char *toHere, const char *fromHere); copies a string from one location to another

size_t strlen(const char *); finds the length of a C string

char *strstr(const char *haystack, const char *needle); finds the first occurrence of the string "needle" in the longer string "haystack".

String to Integer and Integer to string

The functions atoi (ASCII to Integer) and itoa(Integer to ASCII) we can convert a sting containing a number back to an integer and vice versa.

void itoa(int input, char *buffer, int radix)

int atoi (const char *string)

You can also use sprintf which works much the same as the printf function we have been using but stores the result to a string.

sprintf(char * str, const char * format[,...])

%[flags][width][.precision][length]specifier

specifier	Output	Example
c	Character	a
d or i	Signed decimal integer	392
e	Scientific notation (mantise/exponent) using e character	3.9265e+2
E	Scientific notation (mantise/exponent) using E character	3.9265E+2
f	Decimal floating point	392.65
g	Use the shorter of %e or %f	392.65
G	Use the shorter of %E or %f	392.65
o	Signed octal	610
s	String of characters	sample
u	Unsigned decimal integer	7235
x	Unsigned hexadecimal integer	7fa
X	Unsigned hexadecimal integer (capital letters)	7FA
p	Pointer address	B800:0000

Source: ("sprintf - C++ Reference", <http://www.cplusplus.com/reference/cstdio/sprintf.html>, 25-8-08)