NUS Library Documentation

Version 1

Developed and Written by Kim Pampusch

1 Introduction

NUS library is video game development library written in c.

2 Minimum Specs

Requires os: Linux or Windows
Requires video driver supporting Vulkan 1.0.0 or later

3 Development

3.1 Development Environments

3.1.1 Linux

3.1.1.1 Debian

Copy the following into a terminal

```
cd ~
git clone https://github.com/PyCee/NUS_library.git
cd NUS_library/
make install
make recompile
```

3.1.1.2 Other
idk

3.1.2 Windows

Idk

3.2 Debug

To enable debugging, the library and your application must both be compiled with the macro

```
NUS_DEBUG
```

This enables Vulkan validation layers and debug extension, both of which must be visible to the application upon compilation.

3.3 Unit Tests

Once the environment is setup, the developer may run the unit tests located in the directories in NUS_library/unit_tests/ to test library functionality.
All tests may be run with the following commands:

```
cd ~/NUS_library/unit_tests/
make recompile
make run
```

4 Error Checking

A function that supports error checking will return a NUS_result

```
typedef enum NUS_result{
  NUS_FAILURE = 0,
  NUS_SUCCESS = 1
} NUS_result;
```

4.1 Results

4.1.1 Failure

NUS_FAILURE means the called function failed in such a way that the program must terminate.

4.1.2 Success

NUS_SUCCESS represents a complete success with no cause for worry

5 Strings

5.1 Absolute Path

An absolute path can be represented by the following structure.

```
#define NUS_ABSOLUTE_PATH_MAX_STRING_COUNT 100
typedef struct NUS_absolute_path{
  char path[NUS_ABSOLUTE_PATH_MAX_STRING_COUNT];
} NUS_absolute_path;
```

```
NUS_absolute_path nus_absolute_path_build(char const * const relative_path)
```

PARAMETERS

relative_path - a nul-terminated c-string that represents the path to a file relative to the final executable

DESCRIPTION

Returns a structure that contains relative_path prefixed by the absolute path to the executable

5.2 String Group

A structure to manage a group of strings.

NUS_string_group

```
void nus_string_group_build(NUS_string_group *p_string_group)
```

PARAMETERS

p_string_group - a pointer to an uninitialized or freed string group

DESCRIPTION

Initializes the string group pointed to by p_string_group

The string group pointed to by p_string_group has a string count of 0

```
void nus_string_group_free(NUS_string_group *p_string_group)
```

PARAMETERS

p_string_group - a pointer to an initialized string group

DESCRIPTION

Frees the string group pointed to by p_string_group

```
void nus_string_group_append(NUS_string_group *p_string_group, const char *string)
```

PARAMETERS

p_string_group - a pointer to an initialized string group string - a null-terminated c-string

DESCRIPTION

Appends a copy of string to the string group pointed to by p_string_group
Increments the string count of the string group pointed to by p_string_group

---

```
NUS_result nus_string_group_set(NUS_string_group *p_string_group,
                                unsigned int index, const char * const string)
```

PARAMETERS

p_string_group - a pointer to an initialized string group
index - the index of the string to set
string - a null-terminated c-string

DESCRIPTION

Sets the string at index of the string group pointed to by p_string_group to string
index must be less than the string count of the string group pointed to by p_string_group

---

```
void nus_string_group_copy(NUS_string_group *p_string_group_dest,
                           NUS_string_group string_group_src)
```

PARAMETERS

p_string_group_dest - a pointer to an initialized string group
string_group_src - an initialized string group

DESCRIPTION

Appends every string in string_group_src to the string group pointed to by p_string_group_dest

---

```
unsigned int nus_string_group_string_index(NUS_string_group string_group,
                                           const char *string)
```

PARAMETERS

p_string_group - an initialized string group
string - a null-terminated c-string

DESCRIPTION

Returns the index of the string in string_group that is equivalent to string
If there is no string equivalent in string_group, the function will return UINT_MAX

```
void nus_string_group_print(NUS_string_group string_group)
```

PARAMETERS

p_string_group - an initialized string group

DESCRIPTION

Outputs information about string_group

6 Save Files

7 Window Management

This library supports creating a window. A window is required to directly receive user input and directly output to the screen.

```
NUS_result nus_window_build(char *title, unsigned short width,
                            unsigned short height,
                            NUS_window *p_window)
```

PARAMETERS

title - a c-string that will appear in the created window's upper bar
width - width of created window
height - height of created window
p_window - pointer to an uninitialized, allocated NUS_window

DESCRIPTION

Initializes and displays a visible, intractable, os-agnostic window

```
void nus_window_free(NUS_window *p_window)
```

PARAMETERS

p_window - pointer to an initialized NUS_window

DESCRIPTION

Frees memory allocated when window is built
The NUS_window referenced by p_window will be destroyed

```
void nus_window_print(NUS_window window)
```

PARAMETERS

window - an initialized NUS_window

DESCRIPTION

prints out information about window

8 User Input

User Input is handled by a series of callbacks. A single NUS_system_events should be created by the application and populated with application defined callbacks. User input events will be obtained from an initialized NUS_window.

Types of user input can be categorized into one of the following:

```
typedef enum NUS_event_type{
  NUS_EVENT_MIN_VALUE = 1,
  NUS_EVENT_CLOSE_WINDOW = 2,
  NUS_EVENT_KEY_PRESS = 3,
  NUS_EVENT_KEY_RELEASE = 4,
  NUS_EVENT_MOUSE_BUTTON_PRESS = 5,
  NUS_EVENT_MOUSE_BUTTON_RELEASE = 6,
  NUS_EVENT_MOUSE_MOTION = 7,
  NUS_EVENT_MOUSE_SCROLL = 8,
  NUS_EVENT_MAX_VALUE = 9,
} NUS_event_type;
```

Each event type has a several subtypes which specify an exact event to respond to

Ex: a specific key to a NUS_EVENT_KEY_PRESS

```
NUS_result nus_event_handler_build(NUS_event_handler *p_event_handler)
```

PARAMETERS

p_event_handler - pointer to an uninitialized, allocated NUS_event_handler

DESCRIPTION

Initializes p_event_handler

```
void nus_event_handler_free(NUS_event_handler *p_event_handler)
```

PARAMETERS

p_event_handler - pointer to an initialized NUS_event_handler

DESCRIPTION

Frees allocated memory of p_event_handler

```
void nus_event_handler_set(NUS_event_handler *p_event_handler)
```

PARAMETERS

p_event_handler - pointer to an initialized NUS_event_handler

DESCRIPTION

Tells the library what NUS_event_handler is responsible for the various callbacks

---

```
void nus_system_events_handle(NUS_window window)
```

PARAMETERS

window - events from this window will be received and handled

DESCRIPTION

Calls callbacks for events that have not been handled

```
#define nus_event_handler_function_append(event_handler, event_type, group_index, function)
```

PARAMETERS

event-handler - an initialized NUS_event_handler that will receive the callback function

event_type - NUS_event_type that specifies what type of event the callback will respond to

group_index - an event subtype that tells, beyond the event_type, what specific event the callback will respond to

function - callback function

SPECIFICS

if event_type is NUS_EVENT_MOUSE_MOTION, function should be of type

```
void (*function)(float, float)
```

otherwise, function should be of type

```
void (*function)(void)
```

DESCRIPTION

sets up a user input based callback

8.1 Close Window

Represented by NUS_event_type NUS_EVENT_CLOSE_WINDOW
Callbacks of this type are called when the user clicks on the close window button of the gui, typically in the top left or right of the window, characterized by an ‘x’.
The only valid subtype is the value 0, as no real subtype exists.
It is recommended to always have a callback that ends the application, so the user always has a standard exit

8.1.1 Example

```
nus_event_handler_append(event_handler, NUS_EVENT_CLOSE_WINDOW, 0,
                                         close_window_callback);
```

8.2 Keyboard

8.2.1 Key Codes

Key code subtypes refer to which key the callback is bound to, which are:

Key Code	Physical Representation	Key Name
NUS_KEY_ESC	ESC	Escape
NUS_KEY_1	1	One
NUS_KEY_2	2	Two
NUS_KEY_3	3	Three
NUS_KEY_4	4	Four
NUS_KEY_5	5	Five
NUS_KEY_6	6	Six
NUS_KEY_7	7	Seven
NUS_KEY_8	8	Eight
NUS_KEY_9	9	Nine
NUS_KEY_0	0	Zero
NUS_KEY_MINUS	-	Minus
NUS_KEY_EQUALS	=	Equals
NUS_KEY_BACKSPACE	BACKSPACE	Backspace
NUS_KEY_TAB	TAB	Tab
NUS_KEY_Q	q	Q
NUS_KEY_W	w	W
NUS_KEY_E	e	E

NUS_KEY_R	r	R
NUS_KEY_T	t	T
NUS_KEY_Y	y	Y
NUS_KEY_U	u	U
NUS_KEY_I	i	I
NUS_KEY_O	o	O
NUS_KEY_P	p	P
NUS_KEY_LBRACKET	[	Left Bracket
NUS_KEY_RBRACKET	]	Right Bracket
NUS_KEY_ENTER	ENTER	Enter
NUS_KEY_LCTRL	CTRL	Left Control
NUS_KEY_A	a	A
NUS_KEY_S	s	S
NUS_KEY_D	d	D
NUS_KEY_F	f	F
NUS_KEY_G	g	G
NUS_KEY_H	h	H
NUS_KEY_J	j	J
NUS_KEY_K	k	K
NUS_KEY_L	l	L
NUS_KEY_SEMICOLON	;	Semi-colon
NUS_KEY_APOSTROPHE	‘	Apostrophe
NUS_KEY_LSHIFT	SHIFT	Left Shift
NUS_KEY_BACKSLASH	\	Backslash
NUS_KEY_Z	z	Z

NUS_KEY_X	x	X
NUS_KEY_C	c	C
NUS_KEY_V	v	V
NUS_KEY_B	b	B
NUS_KEY_N	n	N
NUS_KEY_M	m	M
NUS_KEY_COMMA	,	Comma
NUS_KEY_PERIOD	.	Period
NUS_KEY_RSHIFT	SHIFT	Right Shift
NUS_KEY_KP_MULTIPLY	*	Star
NUS_KEY_LALT	ALT	Left Alt
NUS_KEY_SPACE		Spacebar
NUS_KEY_NUM_LOCK	NUM LOCK	Num Lock
NUS_KEY_KP_7	7	Keypad Seven
NUS_KEY_KP_8	8	Keypad Eight
NUS_KEY_KP_9	9	Keypad Nine
NUS_KEY_KP_MINUS	-	Keypad Minus
NUS_KEY_KP_4	4	Keypad Four
NUS_KEY_KP_5	5	Keypad Five
NUS_KEY_KP_6	6	Keypad Six
NUS_KEY_KP_PLUS	+	Keypad Plus
NUS_KEY_KP_1	1	Keypad One
NUS_KEY_KP_2	2	Keypad Two
NUS_KEY_KP_3	3	Keypad Three
NUS_KEY_KP_0	0	Keypad Zero

NUS_KEY_KP_PERIOD	.	Keypad Period
NUS_KEY_KP_ENTER	ENTER	Keypad Enter
NUS_KEY_RCTRL	CTRL	Right Control
NUS_KEY_RALT	ALT	Right Alt
NUS_KEY_ARROW_UP	↑	Up Arrow
NUS_KEY_ARROW_LEFT	←	Left Arrow
NUS_KEY_ARROW_RIGHT	→	Right Arrow
NUS_KEY_ARROW_DOWN	↓	Down Arrow

8.2.2 Examples

```
nus_event_handler_append(event_handler, NUS_EVENT_KEY_PRESS,
                         NUS_KEY_W, w_press_callback);
nus_event_handler_append(event_handler, NUS_EVENT_KEY_RELEASE,
                         NUS_KEY_R, r_release_callback);
```

8.3 Mouse

8.3.1 Motion

Mouse motion requires a 0 in place of a subtype. No real subtype exists.
Mouse motion callbacks takes (float, float) as parameters.
The first parameter is the change in the x position of the mouse since the last motion event detected.
The second parameter is that of the y position.

8.3.2 Button

A mouse button has the subtypes:
NUS_MOUSE_BUTTON_LEFT, NUS_MOUSE_BUTTON_RIGHT, and NUS_MOUSE_BUTTON_MIDDLE

8.3.3 Scroll

A scroll event has the subtypes:
NUS_SCROLL_UP, NUS_SCROLL_DOWN, NUS_SCROLL_LEFT, and NUS_SCROLL_RIGHT

8.3.4 Examples

```
nus_event_handler_append(event_handler, NUS_EVENT_MOUSE_MOTION, 0,
                         motion_callback);
nus_event_handler_append(event_handler, NUS_EVENT_MOUSE_SCROLL,
                         NUS_SCROLL_UP, scroll_up_callback);
nus_event_handler_append(event_handler,
                         NUS_EVENT_MOUSE_BUTTON_PRESS,
                         NUS_MOUSE_BUTTON_RIGHT,
                         right_button_press_callback);
```

9 3D Math

9.1 Vectors

A 3D vector contains a x, a y, and a z component. A vector is used to describe a point in a 3D space, or to describe a direction (directions must be normalized).

A 3D vector is represented by a NUS_vector.

```
typedef struct NUS_vector{
 double x, y, z;
} NUS_vector;
```

```
NUS_vector nus_vector_build(double x, double y, double z)
  PARAMETERS
    x - x value of the resulting vector
    y - y value of the resulting vector
    z - z value of the resulting vector
  DESCRIPTION
    Returns an initialized vector with the x, y, and z values of the parameters
```

```
NUS_vector nus_vector_add(NUS_vector vector_0, NUS_vector vector_1)
  PARAMETERS
    vector_0 - the initialized augend
    vector_1 - the initialized addend
  DESCRIPTION
    Returns the sum of the parameters
```

```
NUS_vector nus_vector_scale(NUS_vector vector, double scale)
  PARAMETERS
    vector - the base initialized vector
    scale - scalar value vector will be multiplied by
  DESCRIPTION
    Returns vector scaled by the scalar
```

```
NUS_vector nus_vector_subtract(NUS_vector vector_0, NUS_vector vector_1)
  PARAMETERS
    vector_0 - the initialized minuend
    vector_1 - the initialized subtrahend
  DESCRIPTION
    Returns the second parameter subtracted from the first
```

```
NUS_vector nus_vector_normalize(NUS_vector vector)
```

PARAMETERS

vector - initialized vector to be normalized

DESCRIPTION

Returns a normalized vector

```
double nus_vector_dot(NUS_vector vector_0, NUS_vector vector_1)
```

PARAMETERS

vector_0 - first initialized vector
vector_1 - second initialized vector

DESCRIPTION

Returns dot product of vector_0 and vector_1

```
NUS_vector nus_vector_cross(NUS_vector vector_0, NUS_vector vector_1)
```

PARAMETERS

vector_0 - first initialized vector
vector_1 - second initialized vector

DESCRIPTION

Returns cross product of vector_0 and vector_1

```
NUS_vector nus_vector_interpolate(NUS_vector vector_0, NUS_vector vector_1,
double t)
```

PARAMETERS

vector_0 - first initialized vector
vector_1 - second initialized vector
t - interpolation progress

DESCRIPTION

Returns a vector from vector_0 to vector_1 by t
A t <= 0.0 returns vector_0
A t >= 1.0 returns vector_1
A t of 0.5 returns a vector halfway between vector_0 and vector_1

```
char nus_vector_cmp(NUS_vector vector_0, NUS_vector vector_1, double range)
```

PARAMETERS

vector_0 - first initialized vector
vector_1 - second initialized vector
Range - maximum distance

DESCRIPTION

Returns (range <= distance between vector_0 and vector_1)

---

void nus_vector_print(NUS_vector vector)

PARAMETERS

vector - an initialized vector

DESCRIPTION

Outputs information about vector

9.2 Quaternions

Quaternions are represented by

```
typedef struct NUS_quaternion{
  double w, x, y, z;
} NUS_quaternion;
```

nus_quaternion_unit
nus_quaternion_lerp
nus_quaternion_slerp
nus_quaternion_apply_rotation
nus_quaternion_print

NUS_quaternion nus_quaternion_build(double w, double x, double y, double z)

PARAMETERS

w - w-value
x - x-value
y - y-value
z - z-value

DESCRIPTION

Returns a quaternion with the values w, x, y and z

NUS_quaternion nus_quaternion_pure(NUS_vector vector)

PARAMETERS

vector - an initialized vector

DESCRIPTION

Returns a pure quaternion built from vector

NUS_quaternion nus_quaternion_unit(NUS_vector vector, double radians)

PARAMETERS

vector - an initialized, normalized vector radians - extent of rotation in radians

DESCRIPTION

Returns a unit quaternion built from vector and radians

```
NUS_quaternion nus_quaternion_conjugate(NUS_quaternion quaternion)
```

PARAMETERS

quaternion - an initialized quaternion

DESCRIPTION

Returns the conjugate of quaternion

```
NUS_quaternion nus_quaternion_h_product(NUS_quaternion quaternion_0,
                                         NUS_quaternion quaternion_1)
```

PARAMETERS

quaternion_0 - first initialized quaternion

quaternion_1 - second initialized quaternion

DESCRIPTION

Returns the h product of quaternion_0 and quaternion_1

```
NUS_vector nus_quaternion_apply_rotation(NUS_quaternion quaternion,
                                         NUS_vector vector)
```

PARAMETERS

quaternion - an initialized unit quaternion

vector - an initialized vector

DESCRIPTION

Returns quaternion applied to vector

```
NUS_quaternion nus_quaternion_lerp(NUS_quaternion quaternion_0,
                                   NUS_quaternion quaternion_1, double t)
```

PARAMETERS

quaternion_0 - an initialized unit quaternion

quaternion_1 - an initialized unit quaternion

t - lerp progress

DESCRIPTION

Returns a quaternion lerp by t between quaternion_0 and quaternion_1
If t <= 0.0, returns quaternion_0
If t >= 1.0, returns quaternion_1
Is an approximation. For 100% realistic interpolation, use slerp

---

```
NUS_quaternion nus_quaternion_slerp(NUS_quaternion quaternion_0,
                                                       NUS_quaternion quaternion_1, double t)
```

PARAMETERS

quaternion_0 - an initialized unit quaternion
quaternion_1 - an initialized unit quaternion
t - lerp progress

DESCRIPTION

Returns a quaternion slerp by t between quaternion_0 and quaternion_1
If t <= 0.0, returns quaternion_0
If t >= 1.0, returns quaternion_1

```
NUS_quaternion nus_quaternion_normalize(NUS_quaternion quaternion)
```

PARAMETERS

quaternion - an initialized quaternion

DESCRIPTION

Returns a normalized quaternion

```
void nus_quaternion_print(NUS_quaternion quaternion)
```

PARAMETERS

quaternion - an initialized quaternion

DESCRIPTION

Outputs information about quaternion

9.3 Axes

A 3D rotation can be best described using a set of 3 local axes (plural of “axis”)

```
 typedef struct NUS_axes{
  NUS_vector forward, upward, left;
} NUS_axes;
```

```
NUS_axes nus_axes_build(NUS_vector forward, NUS_vector upward, NUS_vector left)
```

PARAMETERS

forward - an initialized vector
upward- an initialized vector
left- an initialized vector

DESCRIPTION

Returns an initialized set of axes from the parameters

NUS_axes nus_axes_interpolate(NUS_axes axes_0, NUS_axes axes_1, double t)

PARAMETERS

axes_0 - first initialized set of axes
axes_1 - second initialized set of axes
t - interpolation progress

DESCRIPTION

Returns an initialized set of axes interpolated between axes_0 and axes_1
If t <= 0.0, returns axes_0
If t >= 1.0, returns axes_1

NUS_axes nus_axes_invert(NUS_axes axes)

PARAMETERS

axes - an initialized set of axes

DESCRIPTION

Returns an initialized set of axes equivalent to the inverted version of axes

---

void nus_axes_print(NUS_axes axes)

PARAMETERS

axes - an initialized set of axes

DESCRIPTION

Outputs information about axes

NUS_axes nus_axes_local_pitch(NUS_axes axes, double radians)

PARAMETERS

axes - an initialized set of axes
radians - extent of rotation, in radians

DESCRIPTION

Returns axes with all axes rotated around axes.left

NUS_axes nus_axes_local_yaw(NUS_axes axes, double radians)

PARAMETERS

axes - an initialized set of axes
radians - extent of rotation, in radians

DESCRIPTION

Returns axes with all axes rotated around axes.upward

NUS_axes nus_axes_local_roll(NUS_axes axes, double radians)

PARAMETERS

axes - an initialized set of axes

radians - extent of rotation, in radians

DESCRIPTION

Returns axes with all axes rotated around axes.forward

`NUS_axes nus_axes_global_pitch(NUS_axes axes, double radians)`

PARAMETERS

axes - an initialized set of axes

radians - extent of rotation, in radians

DESCRIPTION

Returns axes with all axes rotated around (1.0, 0.0, 0.0)

`NUS_axes nus_axes_global_yaw(NUS_axes axes, double radians)`

PARAMETERS

axes - an initialized set of axes

radians - extent of rotation, in radians

DESCRIPTION

Returns axes with all axes rotated around (0.0, 1.0, 0.0)

`NUS_axes nus_axes_global_roll(NUS_axes axes, double radians)`

PARAMETERS

axes - an initialized set of axes

radians - extent of rotation, in radians

DESCRIPTION

Returns axes with all axes rotated around (0.0, 0.0, 1.0)

`NUS_axes nus_axes_global_rotation(NUS_axes axes, NUS_quaternion quaternion)`

PARAMETERS

axes - an initialized set of axes

quaternion - an initialized unit quaternion

DESCRIPTION

Returns axes with all quaternion applied to each axis

9.4 Matrices

Matrices are represented by

```
typedef struct NUS_matrix{
  float ele[4][4];
} NUS_matrix;
```

```
NUS_matrix nus_matrix_build(float m00, float m01, float m02, float m03,
                            float m10, float m11, float m12, float m13,
                            float m20, float m21, float m22, float m23,
                            float m30, float m31, float m32, float m33)
```

PARAMETERS

each parameter is a matrix value

each parameter is named m[row][column]

DESCRIPTION

Returns a matrix initialized to the values passed as parameters

---

```
NUS_matrix nus_matrix_identity(void)
```

PARAMETERS

No parameters

DESCRIPTION

Returns an initialized identity matrix

```
NUS_matrix nus_matrix_zero(void)
```

PARAMETERS

No parameters

DESCRIPTION

Returns a matrix with all elements initialized to 0.0

```
NUS_matrix nus_matrix_transpose(const NUS_matrix matrix)
```

PARAMETERS

matrix - matrix to be transposed

DESCRIPTION

Returns a transposed version of matrix

```
NUS_matrix nus_matrix_scale(const NUS_matrix matrix, const float s)
```

PARAMETERS

matrix - an initialized matrix

s - a scalar value

DESCRIPTION

Returns matrix scaled by s

```
NUS_matrix nus_matrix_multiply(const NUS_matrix matrix_0, const NUS_matrix
matrix_1)
```

PARAMETERS

matrix_0 - first initialized matrix

matrix_1 - second initialized matrix

DESCRIPTION

Returns matrix_0 * matrix_1 (in that order)

NUS_matrix nus_matrix_transformation(NUS_vector vector, NUS_axes axes)

PARAMETERS

vector - an initialized NUS_vector that represents a translation

axes - an initialized set of axes, representing a 3D rotation

DESCRIPTION

Returns a transformation matrix that rotates to axes then translates by vector

---

NUS_vector nus_matrix_transform(NUS_matrix matrix, NUS_vector vector)

PARAMETERS

matrix - an initialized transformation matrix

vector - an initialized vector

DESCRIPTION

Returns vector transformed by matrix

NUS_matrix nus_matrix_inverted(NUS_matrix matrix)

PARAMETERS

matrix - an initialized matrix

DESCRIPTION

Returns the inverse of matrix

---

void nus_matrix_print(NUS_matrix matrix)

PARAMETERS

matrix - an initialized matrix

DESCRIPTION

Outputs matrix values

10 Vulkan

To work with the computer’s gpu for rendering, Vulkan is used.
From the official Vulkan website https://www.khronos.org/vulkan/
“Vulkan is a new generation graphics and compute API that provides high-efficiency, cross-platform access to modern GPUs used in a wide variety of devices from PCs and consoles to mobile phones and embedded platforms.”

While NUS library abstracts away much of the coding a typical Vulkan program requires, proficient knowledge in Vulkan is still required to make full use of NUS library. The developer will still be interacting with many Vulkan structures and functions.

11 GPU Information

11.1 Initialization

11.2 Queue Info

11.3 Command Buffer

11.4 Queue Submit

12 Presentation Surface

12.1 Render Target

12.2 Presenting

Graphics Pipeline

Shaders

Attachments

Pipeline

Model

Custom Format