HLSL Cheat Sheet

Here is a cheat sheet with all the HLSL commands with basic syntax reminders sorted by version compatibility.

Versions are as follows and are inclusive (which means version 1 to 3 are available to DirectX 9 for example):

- 1.x / 2.x = Pre DirectX 9
- 3.x = DirectX 9.0c
- 4.x = DirectX 10
- 5.x = DirectX 11

Language Syntax

Structure Declaration

```
struct MyStruct
{
    float4 position : POSITION;
    float3 normal : NORMAL;
    float2 texcoord : TEXCOORD0;
}
```

Functions

```
float4 MyFunc(float t, sampler2D s, float2 texcoord)
{
    float4 somevalue;
    ...
    return someValue;
}
```


Flow Control

```
 if(i < value)
 { //...
 }
 else
 { //...
 }
```

```
for(i = 0; i < count ; i++)
{
  //...
}
```

Semantics

Semantics gives instructions of which internals of the GPU are used for reading and writing data:

- `TEXCOORD0` : Data is stored either in the mesh's texture coordinate channel 0, or uses the texcoord interpolator channel 0, can use also other channels such as `TEXCOORD1` . `TEXCOORD2` . `TEXCOORD3` .....
- `COLOR` : Color Channel
- `NORMAL` : Normal Channel
- `TANGENT` : Tangent Channel
- `SV_TARGET` : Used when writing color : the render target used to write.
- `SV_TARGET[0]` : When writing to multiple outputs : the target in the array we need to write

Types

GPU are powerful processing units that are designed to be processing masses of floating-point data. But that does not mean that these units are strictly restricted to floating point. Here is the list of the commonly found types in HLSL

- `float` : 32-bit standard floating point value
- `half` : 16-bit floating point value (with reduced precision)

- `double` : 64-bit floating point value (with increased precision)
- `int` , `uint` : 32-bit integers, with sign or without sign
- `byte` : 8-bit unsigned integer
- `bool` 1-bit boolean value (packed into a 32-bit register though)

Vectors

Each type can be packed into vectors of different length, from 1 to 4, per axis:

- `float` can be packed into `float2` , `float3` or `float4`
- `uint` can be packed into `uint2` , `uint3` , `uint4`
- ....

Matrices

Two-dimensional packing of values : for example `float4x3` , `uint3x2`

Arrays

N-Dimensional lists of values : for example `float[3][2]`

Language Functions

Examples feature values which are the following

- x, y, z : values of any components (float, float2, float3, float4)
- vector : multiple component (float2, float3, float4) required

Basic Math

function	example	description	version
abs	`abs(x)`	Absolute value (per component)	1
acos	`acos(x)`	Arc-Cosine (per component)	1
all	`all(vector)`	Returns if all components are nonzero (boolean)	1
any	`any(vector)`	Returns if any of components are nonzero (boolean)	1
asin	`asin(x)`	Arc-Sine (per component)	1
atan	`atan(x)`	Art-Tangent of X (ranged from - π/2 to π/2 , per component)	1
atan2	`atan2(y,x)`	Arc-Tangent of Y/X (ranged from - π/2 to π/2 , per component)	1
ceil	`ceil(x)`	Returns the next greater integer (per-component)	1
clamp	`clamp(x,min,max)`	Clamps the value within range min,max (per-component)	1
clip	`clip(x)`	Discards the current pixel if x is less than zero (any component)	1
cos	`cos(x)`	Returns the cosine of X (per component)	1
cosh	`cosh(x)`	Returns the hyperbolic cosine of X (per component)	1
cross	`cross(vector, vector)`	Returns the cross product between the two vectors	1
degrees	`degrees(x)`	Converts a radians angle value to degrees	1