* How does the GPU allocation of memory and passing of data mirror that of C++ on the CPU?
* What is a uniform?
* GPU - Data is not transferred directly GPU, it’s read by the OS
* C++ - The actual allocation of physical memory to process is done by OS

Martin talks

* CPU
  + malloc > before filling in array, allocating size
  + initializing type
  + primitive types - int 4
  + elements
  + multiply (size of each ele \* # of elements) = total amt of data needs to be allocated
  + fill in chunk of memory
* GPU
  + send in commands > create buffers
  + bufferID > location in memory
  + genBuffer >
  + B-type, handle (what type of data)
  + Bind associates the ID w/ the type of data
  + allocate > you specify attributes, have a pointer to where data is on CPU
  + api call is going to transfer things from CPU > GPU
* CPU transfers memory space to GPU, GPU doesn't have access to ram only access to its own graphics card memory
* these calls tell GPU where to find info, & what’s there

Passing of Data

Data that is shared between the CPU and GPU must be allocated in both memories, and explicitly copied between them by the program.

C++ needs to be compiled to machine language first, before it can be executed.

GPUs dedicates more transistors to data processing, and from the memory usage point of view, CPU is optimized to have the smallest memory latency possible, while the GPU maximizes memory bandwidth for data transfer.

What happens exactly when this "memory" gets allocated?

**you often use shared objects between the threads.**

* **GPU accesses data that’s allocated thru buffers based on type**
* **C++ using the malloc command arrays of data are allocated in memory based on type**

**A uniform is a global** [**GLSL**](https://www.opengl.org/wiki/GLSL) **variable declared with the "uniform"** [**storage qualifier**](https://www.opengl.org/wiki/Type_Qualifier_(GLSL)#Storage_qualifier)**. These act as parameters that the user of a shader program can pass to that program. They are stored in a** [**program object**](https://www.opengl.org/wiki/GLSL_Objects)**.**

**Uniforms are so named because they do not change from one execution of a shader program to the next within a particular rendering call. This makes them unlike shader stage inputs and outputs, which are often different for each invocation of a program stage.**

**Uniform variables are used to communicate with your vertex or fragment shader from "outside". In your shader you use the uniform qualifier to declare the variable:**

| **uniform float myVariable;** |
| --- |
| ***Vertex of Fragment Shader Source Code*** |

**Uniform variables are read-only and have the same value among all processed vertices. You can only change them within your C++ program.**

**---**

deg > how many times you can differentiate a curve

* when you differentiate the deg becomes 1 less
* cubic > quadratic > when you differentiate
* when you differentitate you lose a control point

Curved Geometry in 1 Variable

* i = control pt > index
* j = order
* i+j = degree

Wiki

* t = parameter [0,1]
* i = specific point
* P = control point
* B(t) gives you explicit formula for each control point
* given t, give you the value of what that function is