Introduction to Computer Graphics and High Performance Computing

Muhammad Mobeen Movania

Associate Professor

Habib University

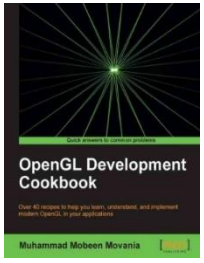
Outlines

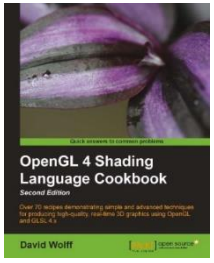
- About the presenter
- What is Computer Graphics, Computer Vision and Digital Image Processing
- Interconnects between CG, CV and DIP
- Applications
- Resources to get started
- Useful libraries and tools
- Our research work in Computer Graphics
- High performance computing using CUDA
- Conclusion

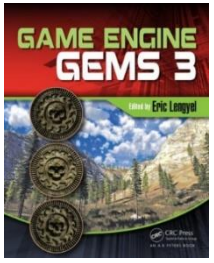
About the Presenter

- PhD, Computer Graphics and Visualization, Nanyang Technological University Singapore, 2012
  - Post-Doc Research at Institute for Infocomm Research (I^2R), A-Star, Singapore (~1.5 years)

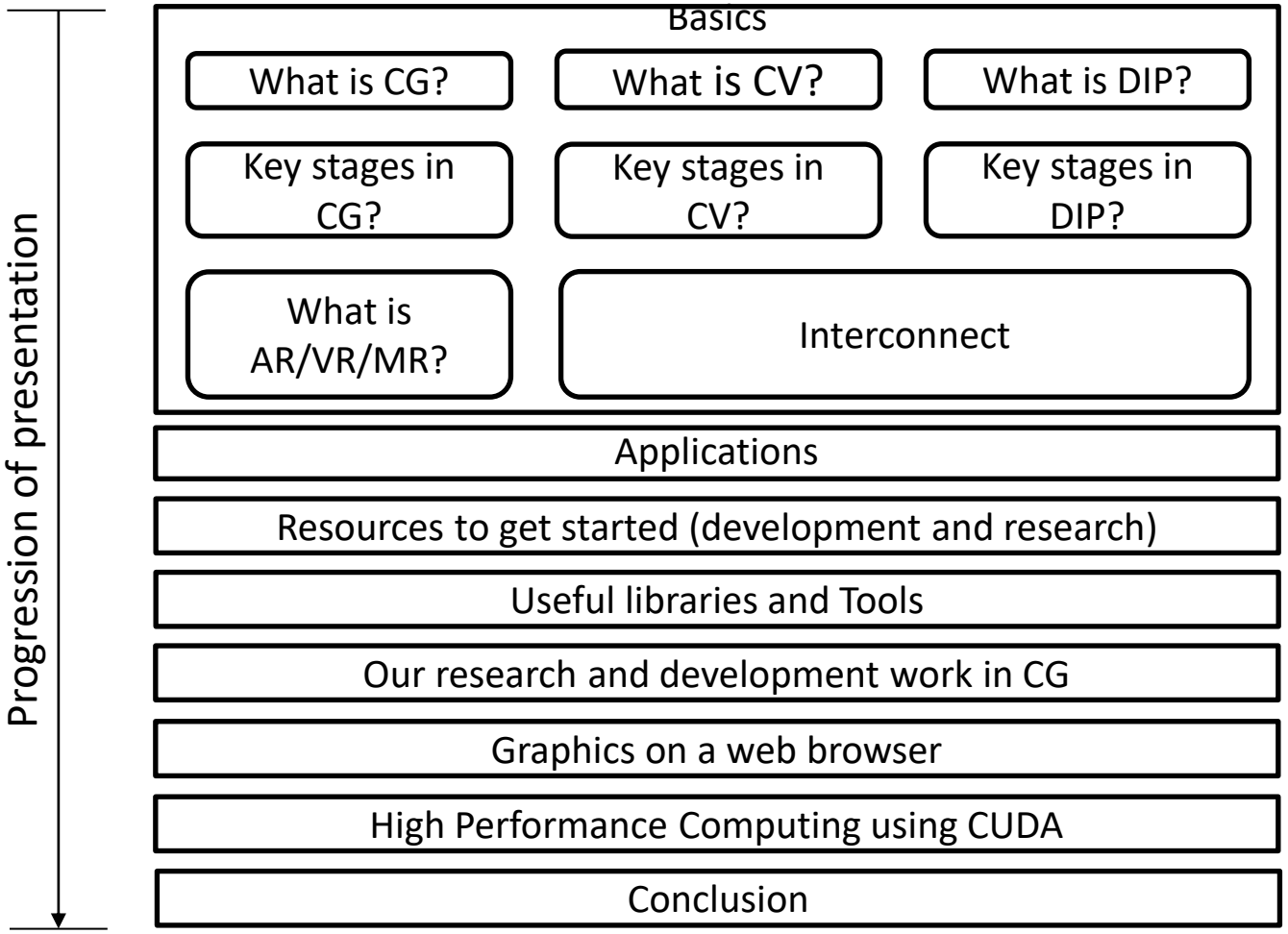
- Publications: 2 books, 4 book chapters, 11 Journals, 13 conferences
  - OpenGL 4 Shading Language Cookbook - Third Edition, 2018. (Reviewer)
  - OpenGL-Build high performance graphics, 2017. (Author)
  - Game Engine Gems 3, April 2016. (Contributor)
  - WebGL Insights, Aug 2015. (Contributor/Reviewer)
  - OpenGL 4 Shading Language Cookbook - Second Edition, 2014. (Reviewer)
  - Building Android Games with OpenGL ES online course, 2014. (Reviewer)
  - OpenGL Development Cookbook, 2013. (Author)
  - OpenGL Insights, 2012. (Contributor/Reviewer)







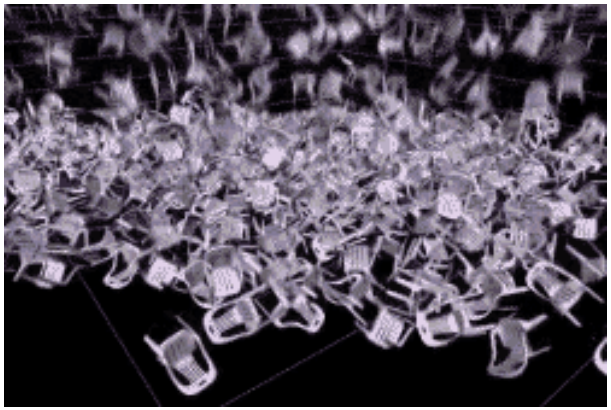
Flow of this Talk

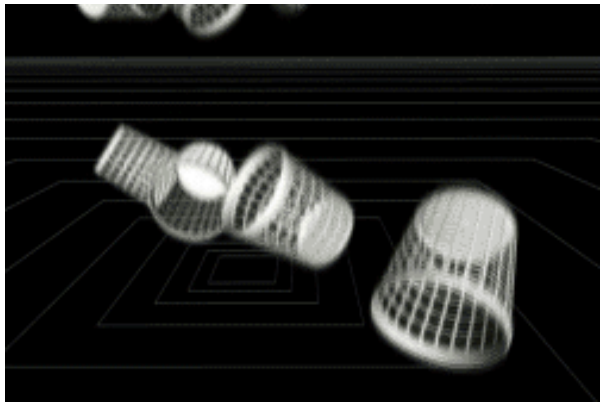


Basics

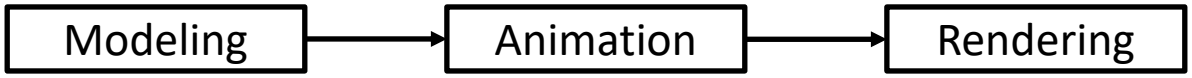
What is Computer Graphics?

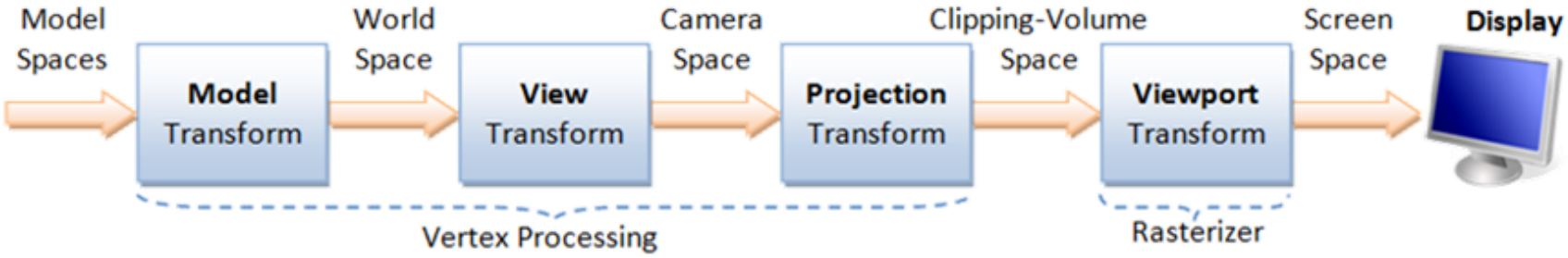
- Converting mathematics into pictures
- Everything that you see on computer screen is either text or graphics
- A powerful medium to communicate
  - A picture is worth a thousand words





Key Stages in Computer Graphics

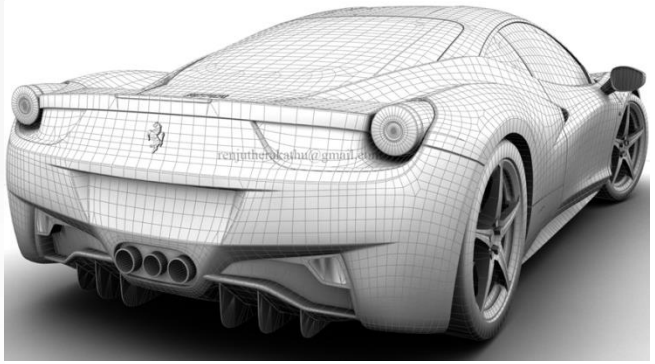


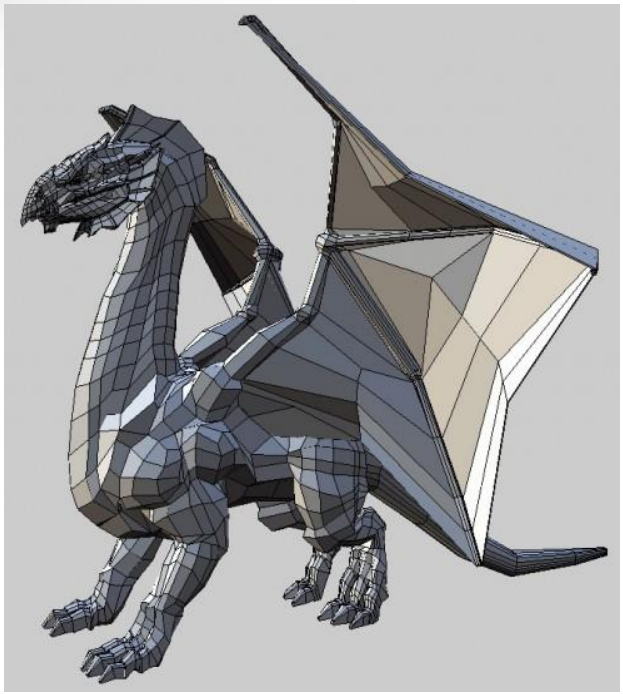


Components of Computer Graphics

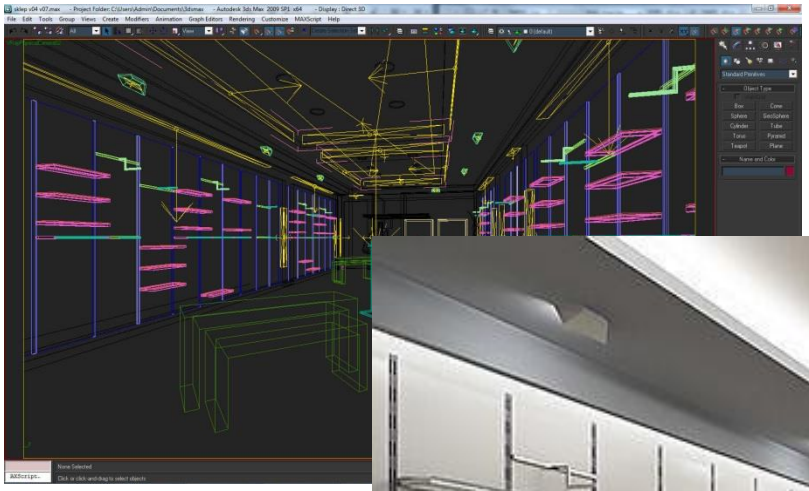
- Three basic components
  - Modeling
  - Rendering
  - Animation
- Modeling
  - Developing a mathematical representation of any 3D *surface* of an object
- Rendering
  - Converting 2D/3D information into pixels
- Animation
  - Changing properties over time to give an illusion of motion

Modeling

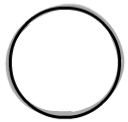




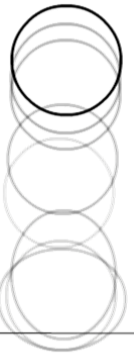
Rendering



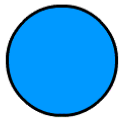
Animation



Frame 1



All Frames



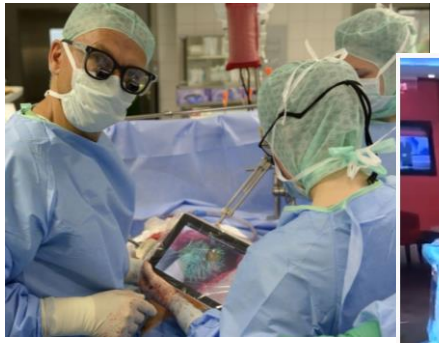
Final Animation

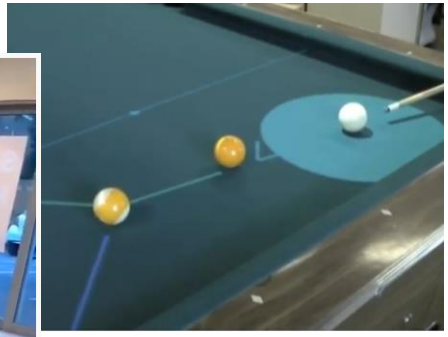
What is Virtual Reality (VR)?

- Experience that **simulates immersive physical presence** in a **real or imagined** environment.

What is Augmented Reality (AR)?

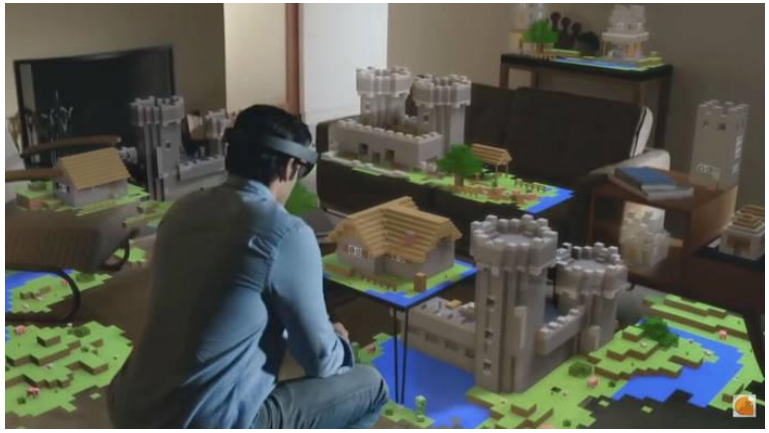
- Experience that **supplements** the **view of a live, physical environment** with **digital** assets.





What is Mixed Reality (MR)?

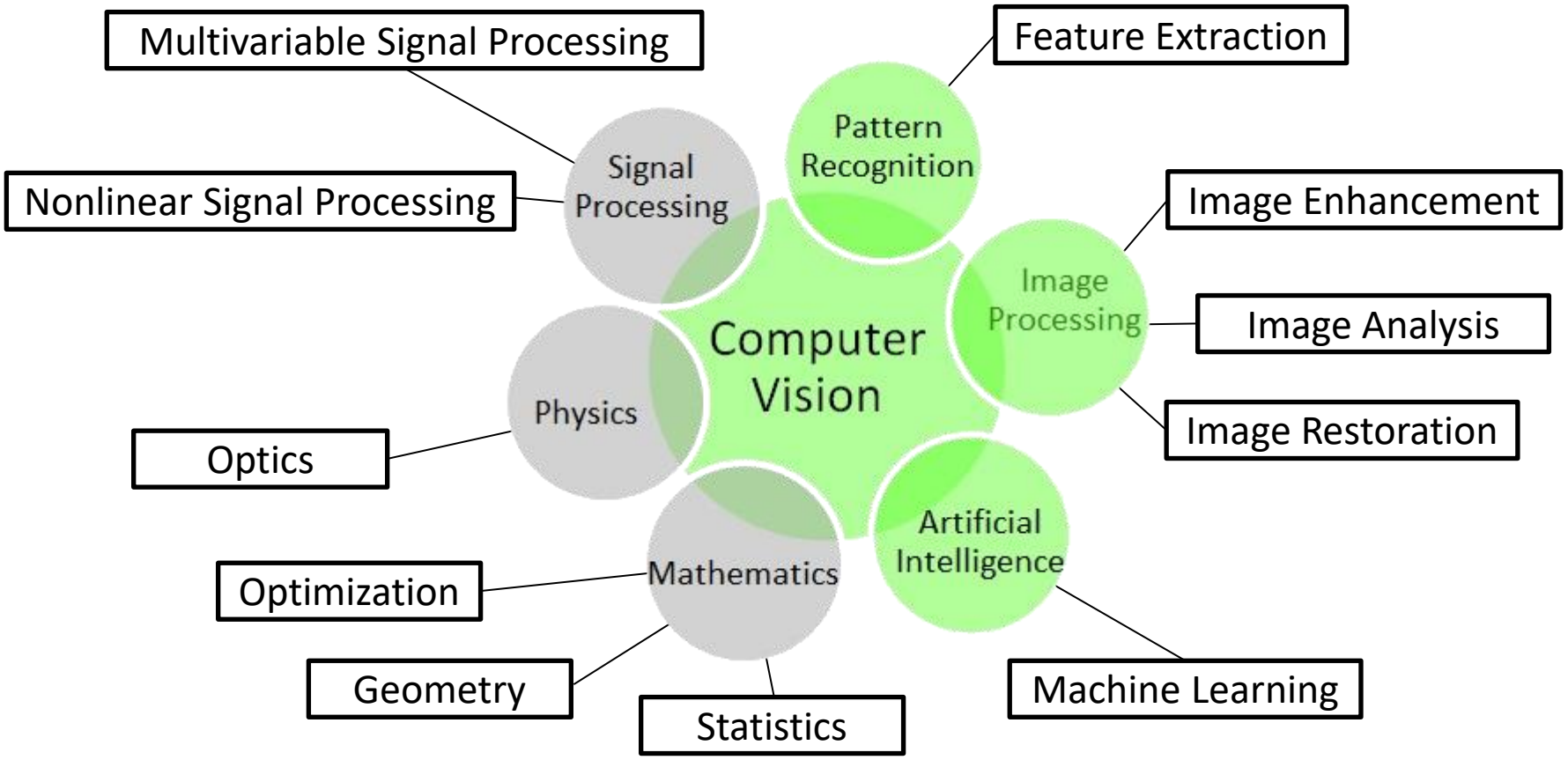
- Mixed reality is overlay of virtual content over the real world but **virtual content and the real-world content** are able to **react to one another** in **real time**.



What is Computer Vision?

Deals with how computers can gain high-level understanding from digital images or videos

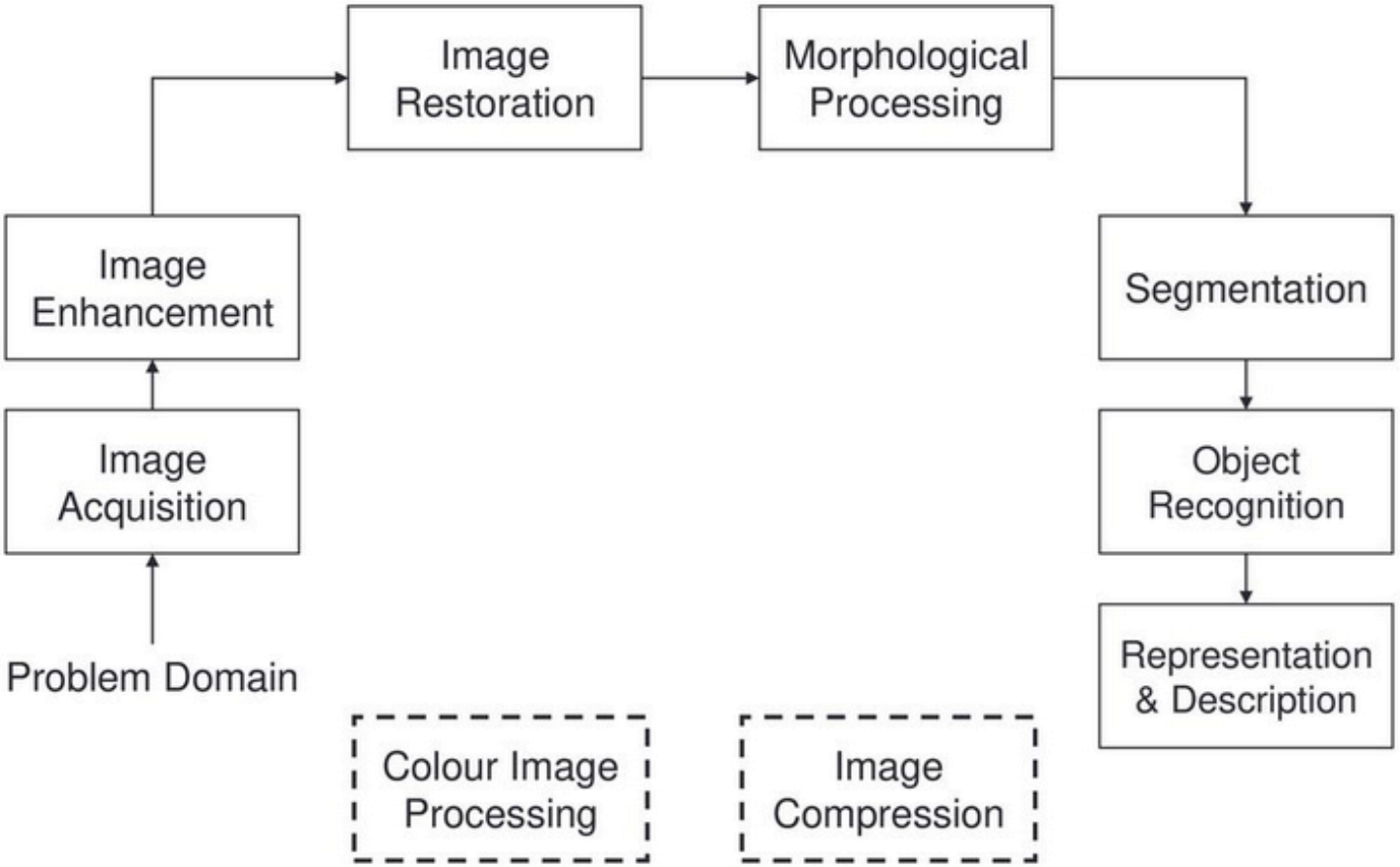
Key stages in Computer Vision



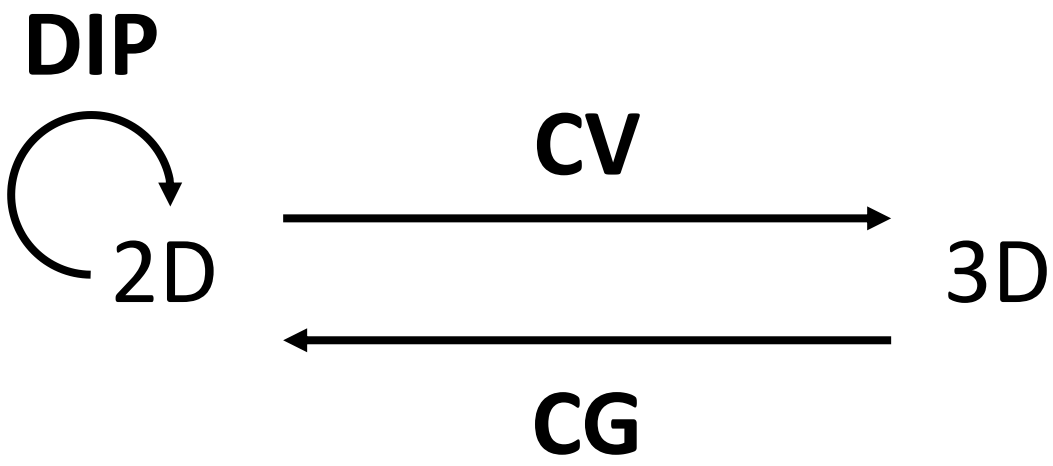
What is (Digital) Image Processing?

Use of a digital computer to process digital images through an algorithm

Key stages in Digital Image Processing



Interconnect - The Crux

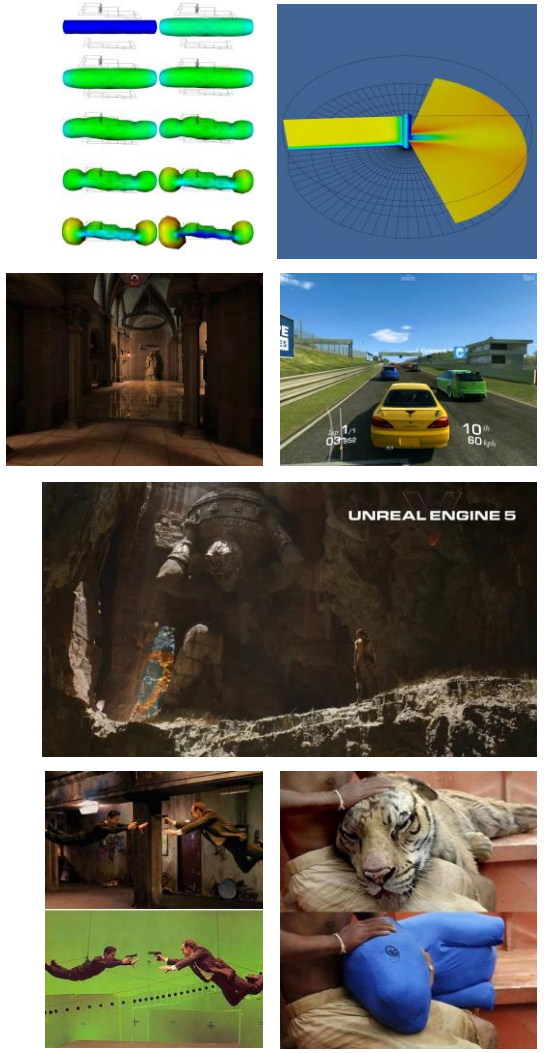


Computer Graphics (CG)
Computer Vision (CV)
Digital Image Processing (DIP)

Applications

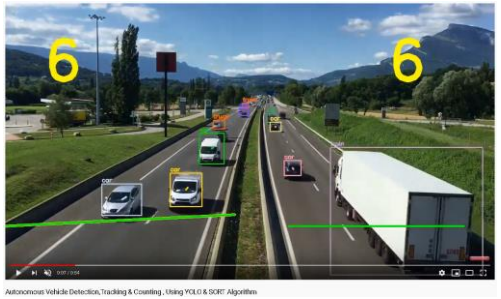
Applications of Computer Graphics

- Applications
  - Scientific visualization
  - Games
  - Movie special effects
  - Tactical trainers and simulators
  - Virtual depiction of real world
  - Archiving digital heritage

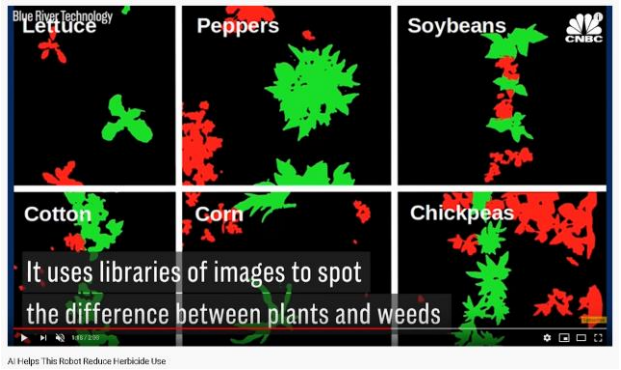


Applications of Computer Vision

Navigation: Self driving cars and drones.



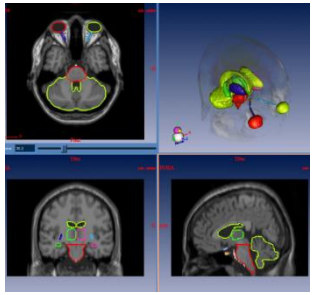
Visual surveillance.



Automatic Crop Identification

Smart Agriculture, disease identification, grading fruit/vegetables,

Applications of Digital Image Processing

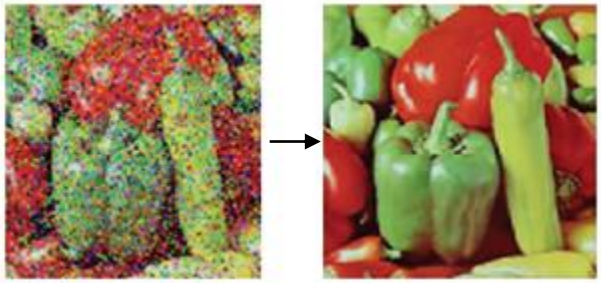


Medical image analysis

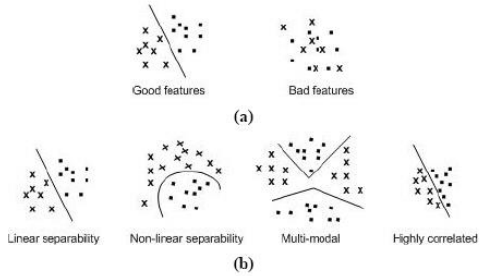
Image restoration

Remote sensing

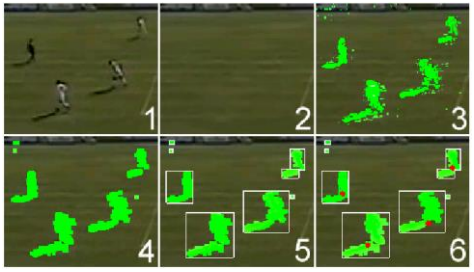
Machine/Robot vision



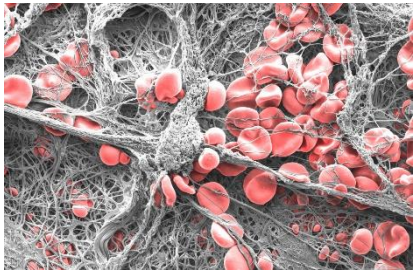
Color image processing



Pattern recognition



Video processing



Microscopic imaging

How to get started in Computer Graphics?

Prerequisites

- Know how to program in C/C++, Python, Matlab
- Strong in data structures and algorithms
- Have a solid understanding of vector mathematics and calculus
- Key courses
  - Calculus and Analytical Geometry
  - Multivariate Calculus
  - Linear Algebra
  - Differential Equations
  - Numerical Computing

Resources for Computer Graphics

Developing graphics applications

- For 2D graphics
  - Three options
    - Make your own engine (software)
    - Use existing game engines (GoDot, Unity, UE4, Cocos2D, etc.)
    - Hardware
      - Platform dependent (using Win32 GDI, GDI+ or Direct2D)
      - Platform independent (using OpenGL,Vulkan)
- For 3D graphics
  - Three options
    - Make your own engine in software
    - Make your own engine using API (DirectX,OpenGL,Vulkan)
    - Use existing engine/framework (Unity, UE4, TwinMotion, Stingray, Lumberyard, OGRE, Irrlicht, CryEngine, Cocos etc.)

Developing Games

- Please don’t start creating your own game engine
  - use an existing game engine otherwise, you will not be able to make a game
- For 2D games
  - Unity,Cocos2D,App Game Kit,Godot,Libgdx, Construct 2,Gamemaker: Studio,Stencyl,Starling
- For 3D games
  - Unity,UE4, TwinMotion, Stingray, CryEngine, Lumberyard

Graphics Frameworks and Libraries

- Graphics API
  - OpenGL/DirectX/Vulkan (for desktop development)
  - OpenGL ES (for mobile/tablets)
  - WebGL (for web browser on desktop/mobile/tablets)
- Shader languages
  - GLSL (OpenGL/OpenGL ES/WebGL)
  - HLSL (DirectX)
  - Cg (wrapper around GLSL for OpenGL applications and HLSL for DirectX applications)
  - Brook (a kernel based GPGPU emulation using shaders)
- Compute
  - CUDA (NVIDIA only)
  - OpenCL (general)
  - WebGPU
  - WebCL (browser based compute)
  - Direct Compute Shader (special shader type in DirectX 10 and above)
  - OpenGL Compute Shader (special shader type in OpenGL v 4.3 and above)
  - Vulkan Compute Shader

Graphics/Game content creation

- 2D Images
  - Photoshop/GIMP/Illustrator
- 3D models
  - 3dsmax/Maya/Blender/Wings3D
- Sound
  - Audacity
- Several online websites having free content
  - https://opengameart.org/
  - http://kenney.nl/assets
  - https://craftpix.net
  - https://v-play.net/game-resources/16-sites-featuring-free-game-sounds
  - https://en.wikipedia.org/wiki/Comparison_of_free_software_for_audio

Computer Graphics Books

- Introductory graphics
  - Real-time Rendering 3rd Edition by Tomas Akenine-Möller, Eric Haines, and Naty Hoffman
  - Fundamentals of Computer Graphics 3rd Edition by Peter Shirley and Steve Marschner
  - Foundations of 3D Computer Graphics by Steven J. Gortler
  - Computer Graphics-Principles and Practice 3rd Edition by John F. Hughes, Andries van Dam, Morgan McGuire, David F. Sklar, James D. Foley, Steven K. Feiner, Kurt Akeley
  - Introduction to 3D Game Programming with DirectX 11 by Frank Luna

- Advanced graphics
  - OpenGL Superbible 6th Edition by Graham Sellers, Richard S Wright and Nicholas Haemel
  - OpenGL 4 Shading Language Cookbook Volume 1 and Volume 2 by David Wolff.
  - OpenGL Development Cookbook by Muhammad Mobeen Movania.
  - OpenGL Insights by Patrick Cozzi and Christophe Riccio
  - GPU Pro Series by Wolfgang Engel
  - GPU Gems Series (Volume 1 to Volume 3)

Computer Graphics Books

- Mathematics and Physics
  - Physics based Animation by Kenny Erleben et al.
  - Game Physics 2nd Edition by David H. Eberly.
  - Geometric Tools for Computer Graphics (The Morgan Kaufmann Series in Computer Graphics) by Philip Schneider, David H. Eberly
  - Game Physics Pearls by Gino van den Bergen and Dirk Gregorius.
  - Mathematics for 3D Game Programming and Computer Graphics 3rd edition by Eric Lengyel.
  - 3D Math Primer for Graphics and Games 2nd edition by Fletcher Dunn and Ian Parberry
  - Real-time collision detection by Christer Ericson

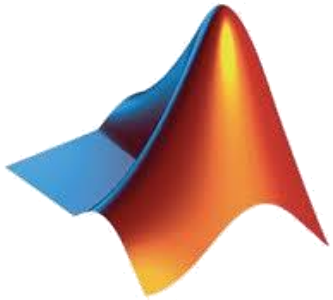
Computer Graphics Books

- Raytracing/PBR/Rendering
  - Physically based Rendering from theory to implementation 3rd Volume by Matt Phar and Greg Humphreys
  - Ray tracing from the ground up by Kevin Suffern Volume 1 and Volume 2 by Kevin Suffern
  - Raytracing in one weekend series by Peter Shirley
  - Realistic Ray Tracing 1 by Peter Shirley and R. Keith Morley
  - Realistic Ray Tracing 2 by Peter Shirley and R. Keith Morley
  - Realistic Image Synthesis using Photon Mapping by Henrik Wann Jensen
  - Principles of Digital Image Synthesis Volume I and Volume II by Andrew Glassner
  - Geometric Tools for Computer Graphics (The Morgan Kaufmann Series in Computer Graphics) by Philip Schneider, David H. Eberly

Resources for Computer Vision and Image Processing

Computer Vision Libraries/Tools

- OpenCV
- Matlab
- ITK
- Aforge.net
- Tensorflow
- SimpleCV
- ImageJ
- scikit-image



Computer Vision/Image Processing Books

- Computer Vision
  - Computer Vision: Algorithms and Applications by Richard Szeliski, 2010.
  - Computer Vision: Models, Learning, and Inference by Simon Prince, 2012.
  - Computer Vision: A Modern Approach by David Forsyth and Jean Ponce, 2002.
  - Introductory Techniques for 3-D Computer Vision by Emanuele Trucco and Alessandro Verri, 1998.
  - Multiple View Geometry in Computer Vision by Richard Hartley and Andrew Zisserman, 2004.
- Image Processing
  - Digital Image Processing by Rafael C Gonzalez and Richard E Woods
  - Image Processing, Analysis and Machine Vision by Milan Sonka and Vaclav Hlavac and Roger Boyle
  - Fundamentals of Digital Image Processing by Anil K Jain

Research Information:
Top journals, conferences and research groups

Graphics Research

- Top journals
  - ACM TOG
  - IEEE TVCG
  - Comp. Graphics Forum
  - IEEE CGA
  - The Visual Computer (TVC)

- Notable books
  - GPU Gems 1,2,3
  - GPU Pro 1,2,3,4,5,6,7,8
  - Game Engine Gems 1,2,3
  - Graphics Gems 1,2,3,4,5,6,7,8
  - PBRT v3/v4 (pbrt.org)
  - http://www.realtimerendering.com/books.html

Top conferences

- SIGGRAPH
- SIGGRAPH Asia
- Eurographics
- IEEE Visualization
- EuroVis
- Pacific Graphics
- HPG
- I3D
- WSCG

Computer Vision/Image Processing Research

- Top Journals
  - IEEE Trans. Pattern Analysis and Machine Intelligence (PAMI)
  - IEEE Trans. Image Processing (TIP)
  - IEEE Trans. Medical Imaging (TMI)
  - Pattern Recognition
  - Int. Journal of Comp. Vision
- Top Conferences
  - IEEE CVPR
  - IEEE ICCV
  - ECCV
  - IEEE ICIP
  - MICCAI
  - IROS
  - WACV

Some famous Graphics Researchers

- http://graphics.pixar.com/library/indexByTitle.html
- http://graphics.stanford.edu/people.html
- http://physbam.stanford.edu/~fedkiw/
- http://www.cs.cornell.edu/~djames/research/index.html
- http://www-bcf.usc.edu/~jbarbic/
- http://graphics.cs.cmu.edu/
- http://graphics.berkeley.edu/
- http://graphics.ethz.ch/
- http://www.cs.ubc.ca/~rbridson/
- http://www.matthiasmueller.info/
- https://cs.uwaterloo.ca/~c2batty/
- http://www.tessendorf.org/reports.html
- https://sites.google.com/site/takahiroharada/
- http://www.physicsbasedanimation.com/
- http://www.math.zju.edu.cn/ligangliu/CAGD/

Notable Local Research Groups[1]

(Computer Graphics/Computer Vision/Image Processing)

- Machine Vision and Learning Lab (SEECS)
- Vision and Image Processing (VIP) UMT
- Computer Vision and Machine Learning Lab (CVML) KICS UET Lahore
- Image and Video Processing Research Group (IVPRG) Comsats
- NEDUET, SmartCity Lab
- ReVeal (Recognition, Vision and Learning) Research Group (FAST NU Ismd)
- The Machine Intelligence Group (FAST NU Khi)

1. The list is non-exhaustive so it might not list all active local research groups

General Tools and Libraries

Some useful libraries/tools

- Processing (https://processing.org)
  - A very simple API written on top of java to create RAD graphics, computer vision and image processing applications
  - Supports both 2D and 3D graphics through OpenGL
  - Highly recommended for beginners

- dlib (http://dlib.net/)
  - A modern C++ toolkit containing machine learning algorithms and tools for creating complex software in C++ to solve real world problems

Some useful libraries/tools

- VTK (https://vtk.org/)
  - a powerful modular open source framework for 3D computer graphics, modeling, image processing, volume rendering, scientific visualization, and 2D plotting
  - allows fast integration and testing of new algorithms and development of robust applications

- ITK (https://itk.org)
  - a powerful modular open source image processing, image segmentation and image registration library

Some useful libraries/tools

- MeVis Lab (http://www.mevislab.de/)
  - a powerful modular framework for image processing research and development with a special focus on medical imaging
  - allows fast integration and testing of new algorithms and the development of clinical application prototypes

- Cinder (http://libcinder.org)
  - Cinder is a free and open source library for professional-quality creative coding in C++
  - Similar to processing but for C++

Some useful libraries/tools

- Paraview (http://www.paraview.org/)
  - a powerful open-source, multi-platform data analysis and visualization application
  - allows users to quickly build visualizations to analyze data

- Meshlab (https://www.meshlab.net/)
  - open source system for processing and editing 3D triangular meshes
  - provides a set of tools for editing, cleaning, healing, inspecting, rendering, texturing and converting meshes.

Graphics on the web browser

Graphics on the Web Browser

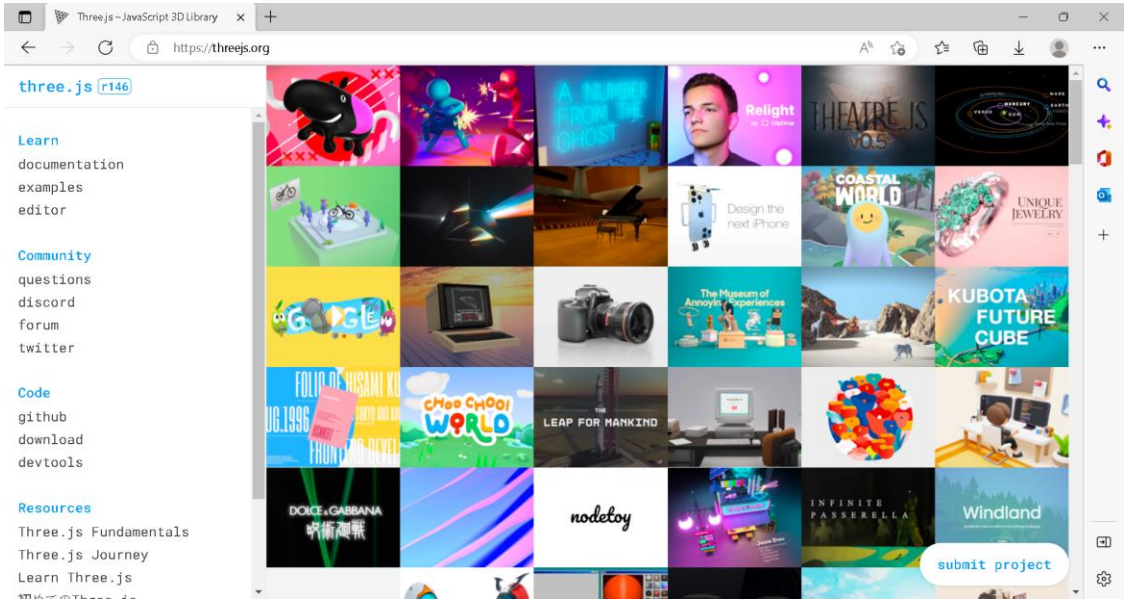
- Possible through WebGL
- Supported through the html5 canvas element
- Low level access to the hardware
- Shader code can be written to access low level hardware features.
- No install required, the web browser needs to support WebGL.
- Supported on most modern desktops, laptops, tablets and smartphones.

So should I start making everything in WebGL?

- It usually depends on the nature of the application you are focusing on
  - If you are working on a demo for your paper, sure.
  - If you are working on a bigger application, don't.
- You should always try to find an existing framework or library
- Many options exists including
  - Three.js
  - Babylon.js
  - PlayCanvas

Three.js

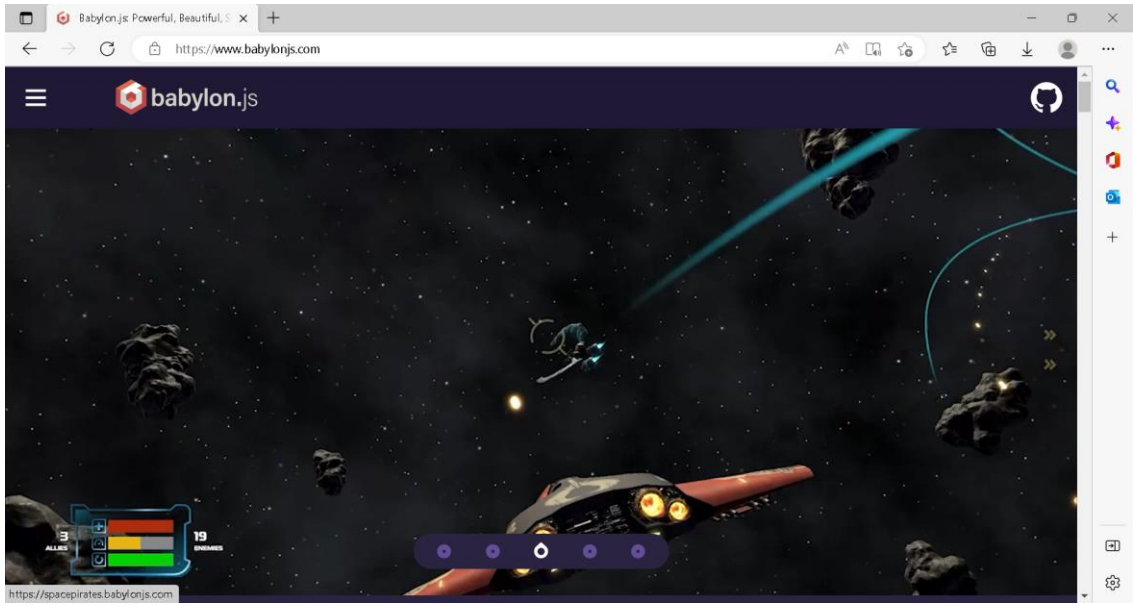
- A powerful scene graph API for web browser.
- Provides a large set of features , extensive documentation, a large user community



https://threejs.org/

Babylon.js

- Similar library to three.js, strong user base
- Not as commonly used as three.js
- Provides access to recent WebGPU standard



https://www.babylonjs.com/

Basics

So how do I get started?

- Both three.js and babylon.js engines provide a large collection of examples.
  - Try to find an example that is similar to your requirement and add on top of it.
  - If you are stuck, google search or stackoverflow will come to your rescue ☺.
- If you like to do things visually
  - Use the three.js editor for basic scenes.
  - Use the babylon.js playground for basic scenes.

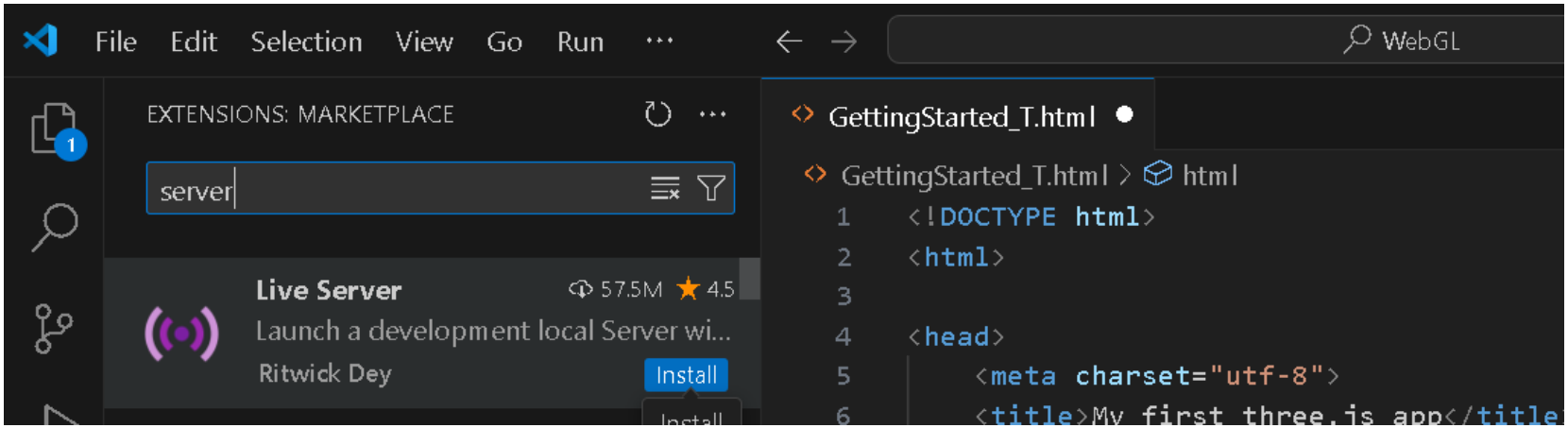
Installation steps

- For node.js
  - `npm install three`
- Then use the import statement in your js code
  - `import * as THREE from 'three';`

- You can also install from CDN or static hosting
  - Details are here: Installation – three.js docs (threejs.org)
  - `<script src=https://unpkg.com/three@0.171.0/build/three.js></script>`

Setting up Visual Studio Code

- Get the live server extension



Setting up Visual Studio Code

- Once live server is installed, select the webpage you want to load and then click “Go live” on the bottom right.
  - This will launch a local server so you can view your application.

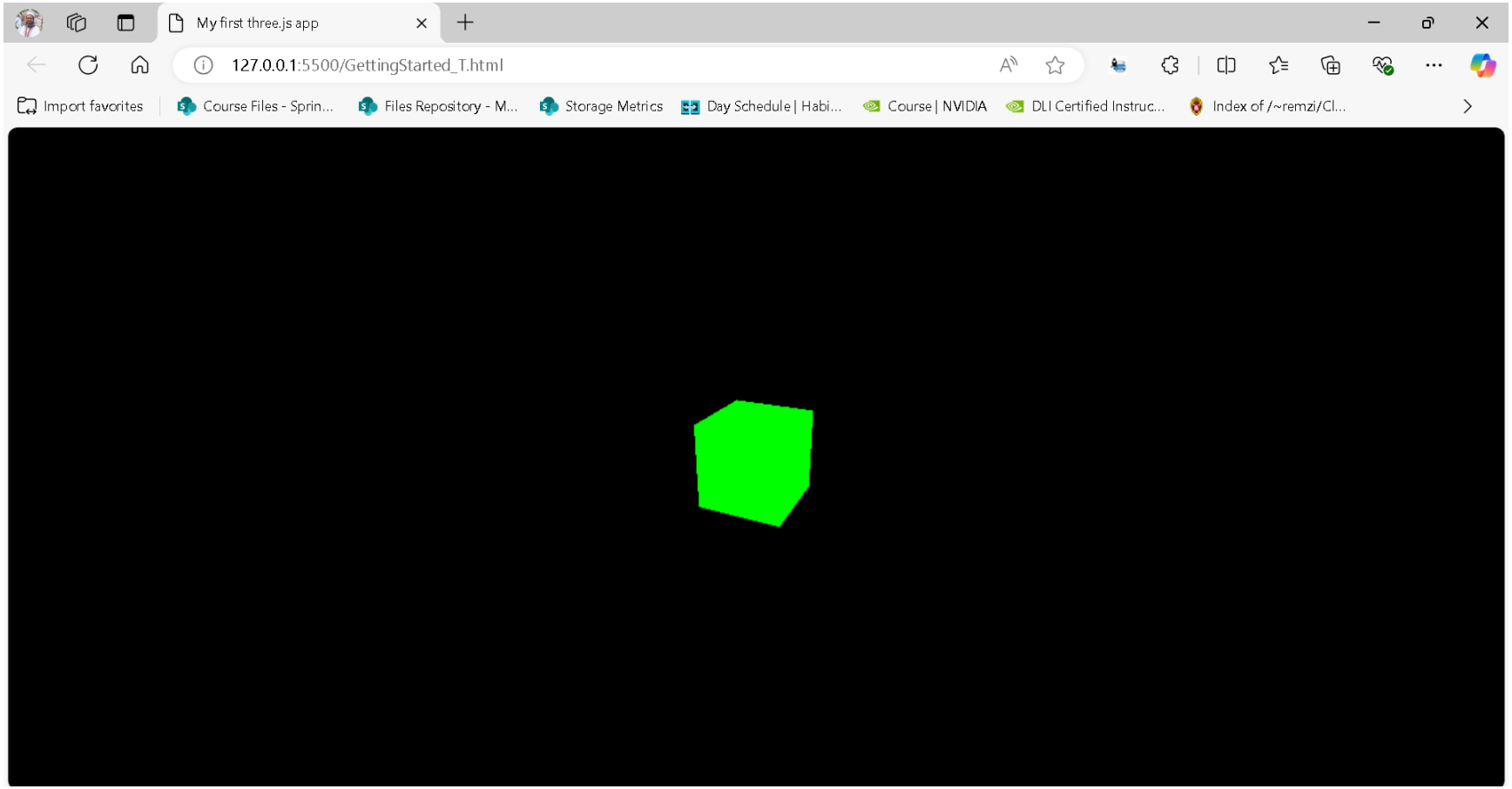
Setting up Visual Studio Code

- Once live server is installed, select the webpage you want to load and then click “Go live” on the bottom right.
  - This will launch a local server so you can view your application.

Basic Example (Three.js)

- Setup is simple
  - Create a THREE.Scene object
  - Create a THREE.Camera object
  - Create your THREE.Mesh object
  - Create a THREE.Renderer object
  - Call render function on the renderer object

Hello World for Three.js

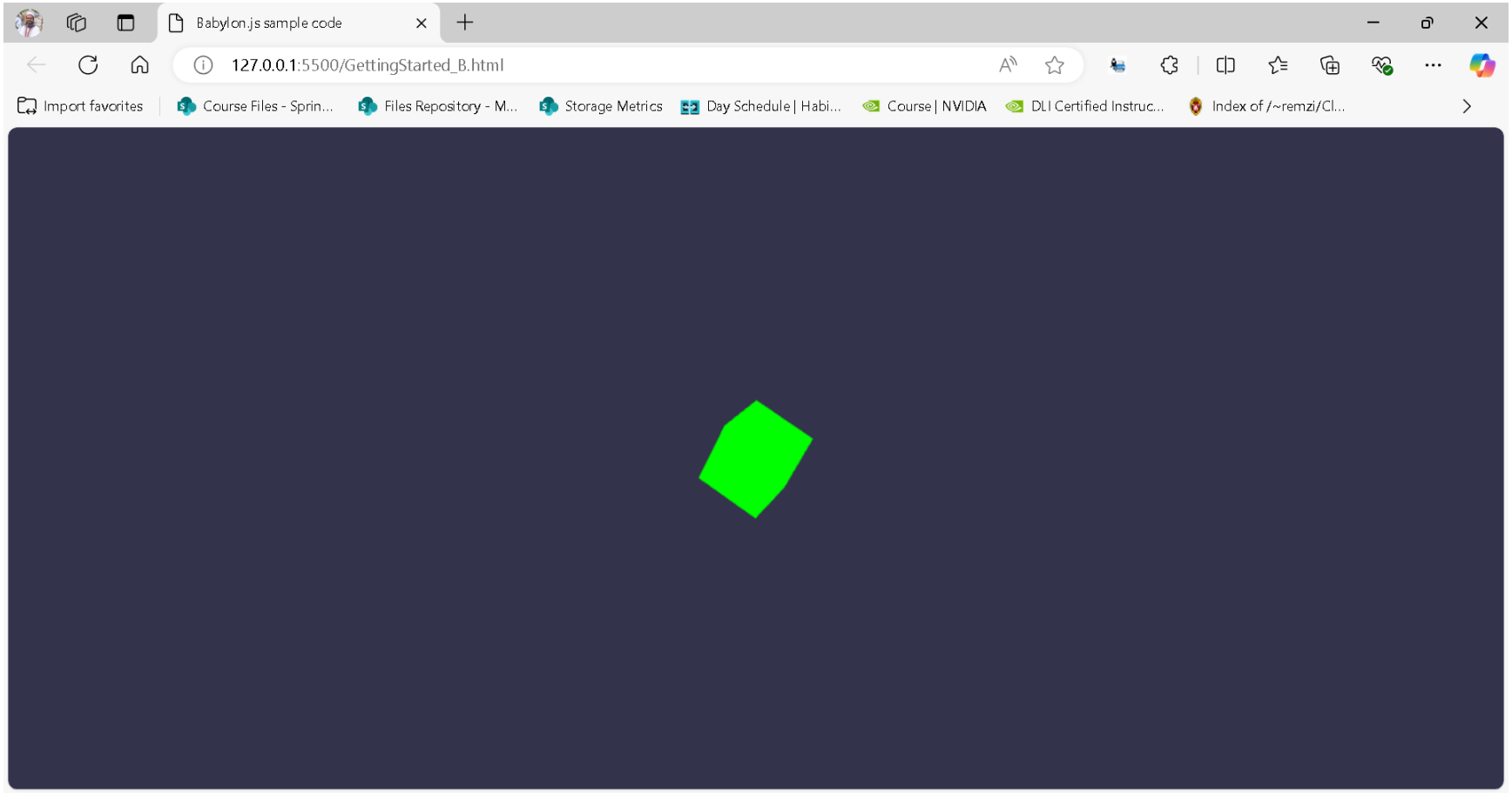


GettingStarted_T.html

Basic Example (Babylon.js)

- Setup is simple
  - Implement a createScene function
    - Create a BABYLON.Scene object
    - Create a BABYLON.FreeCamera object
    - Create your mesh using BABYLON.MeshBuilder
    - Return your scene from the createScene function

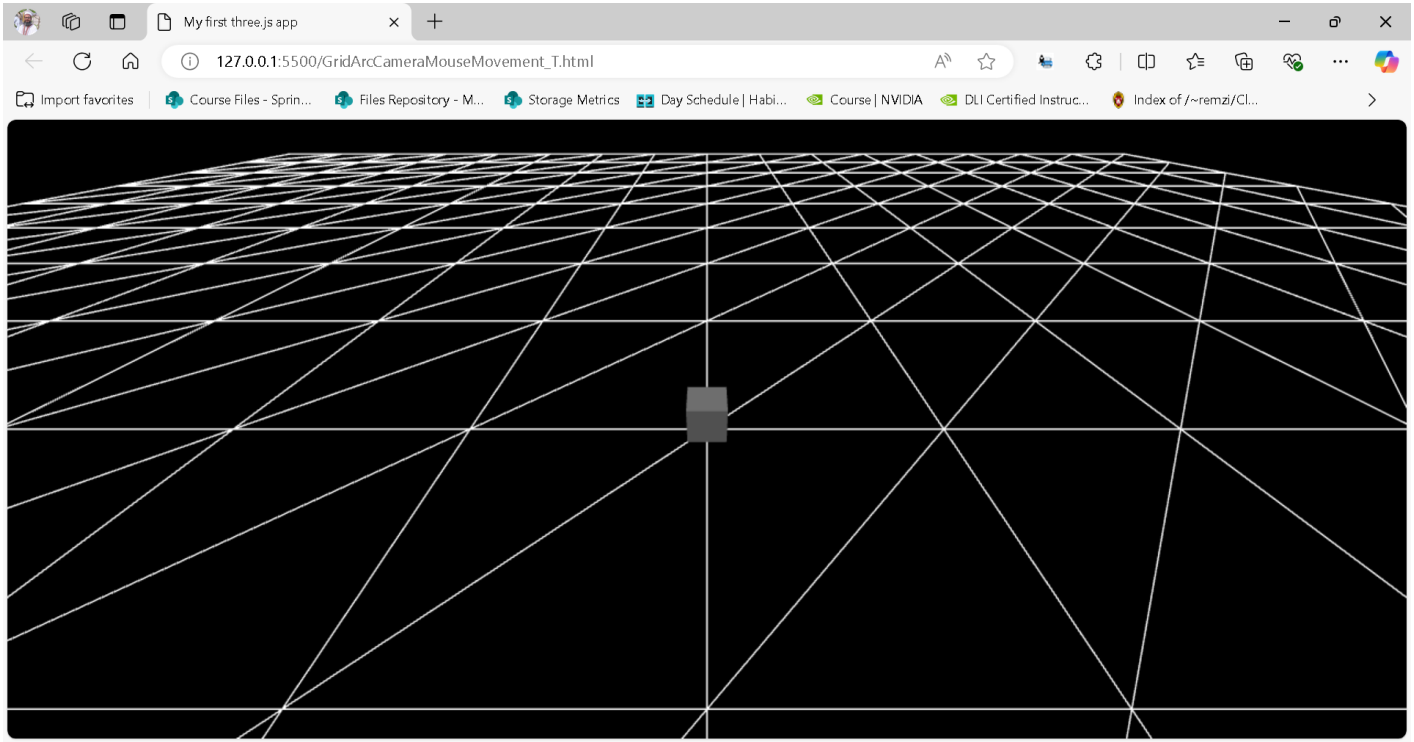
Hello World for Babylon.js



https://playground.babylonjs.com/#T1TB70

More Examples

- Grid with Arc Rotate Camera



GridArcCameraMouseMovement_B.html
GridArcCameraMouseMovement_T.html

How do I load images?

- Both three.js and Babylon.js provide image loaders for a large list of image formats including jpg, gif, png, tiff, tga, dds etc.
- Refer to documentation
  - Three.js
    - Texture – three.js docs (threejs.org)
  - Babylon.js
    - https://doc.babylonjs.com/features/featuresDeepDive/materials/using/materials_introduction#texture

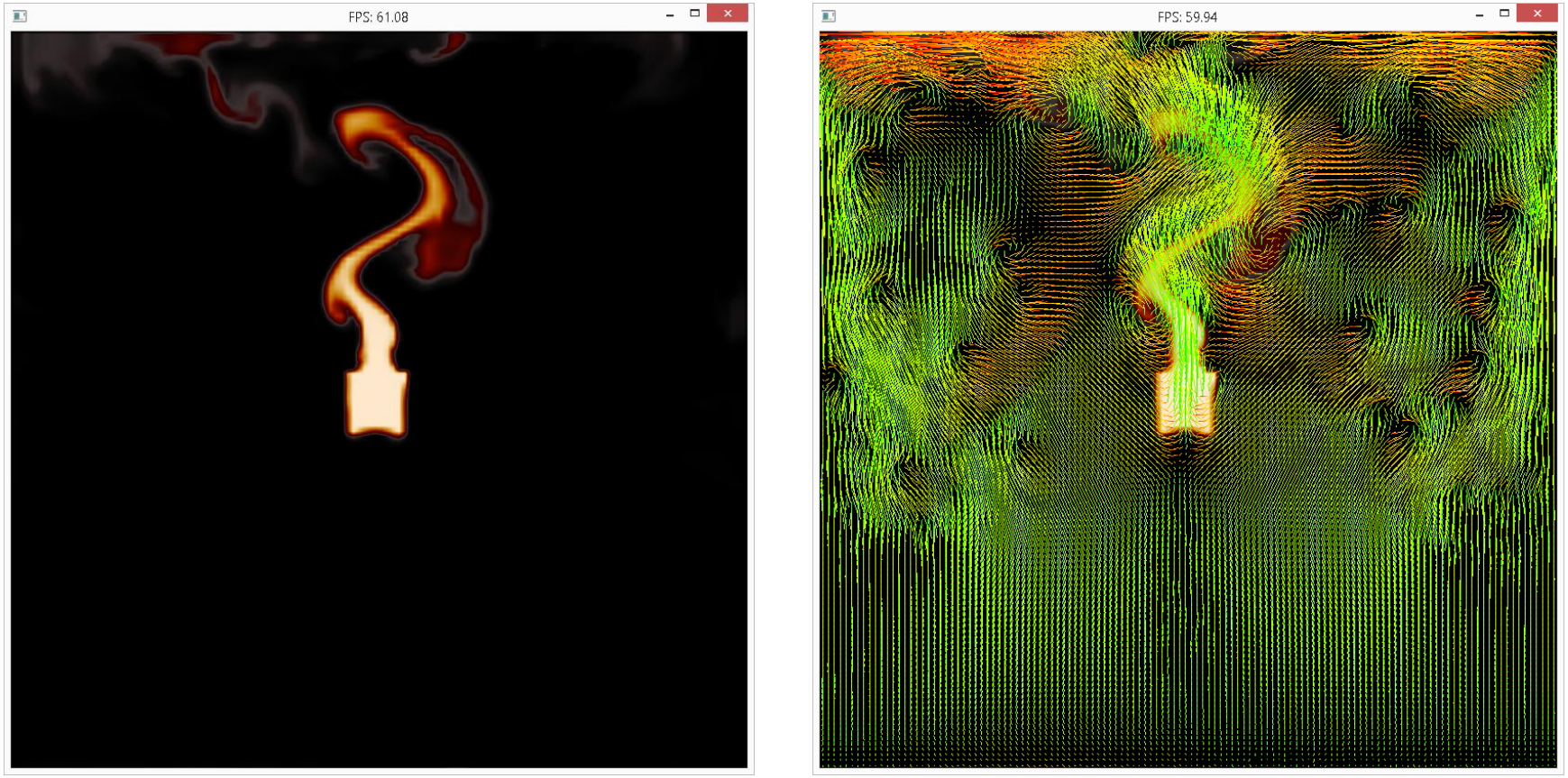
How do I load my own meshes?

- Both three.js and Babylon.js provide support for loading a large list of model formats including 3ds, obj, fbx, gltf, ply, glb, etc.
- Refer to documentation
  - Three.js:
    - https://threejs.org/docs/#manual/en/introduction/Loading-3D-models
  - Babylon.js:
    - https://doc.babylonjs.com/features/featuresDeepDive/importers

Our Research and Development Work in Computer Graphics

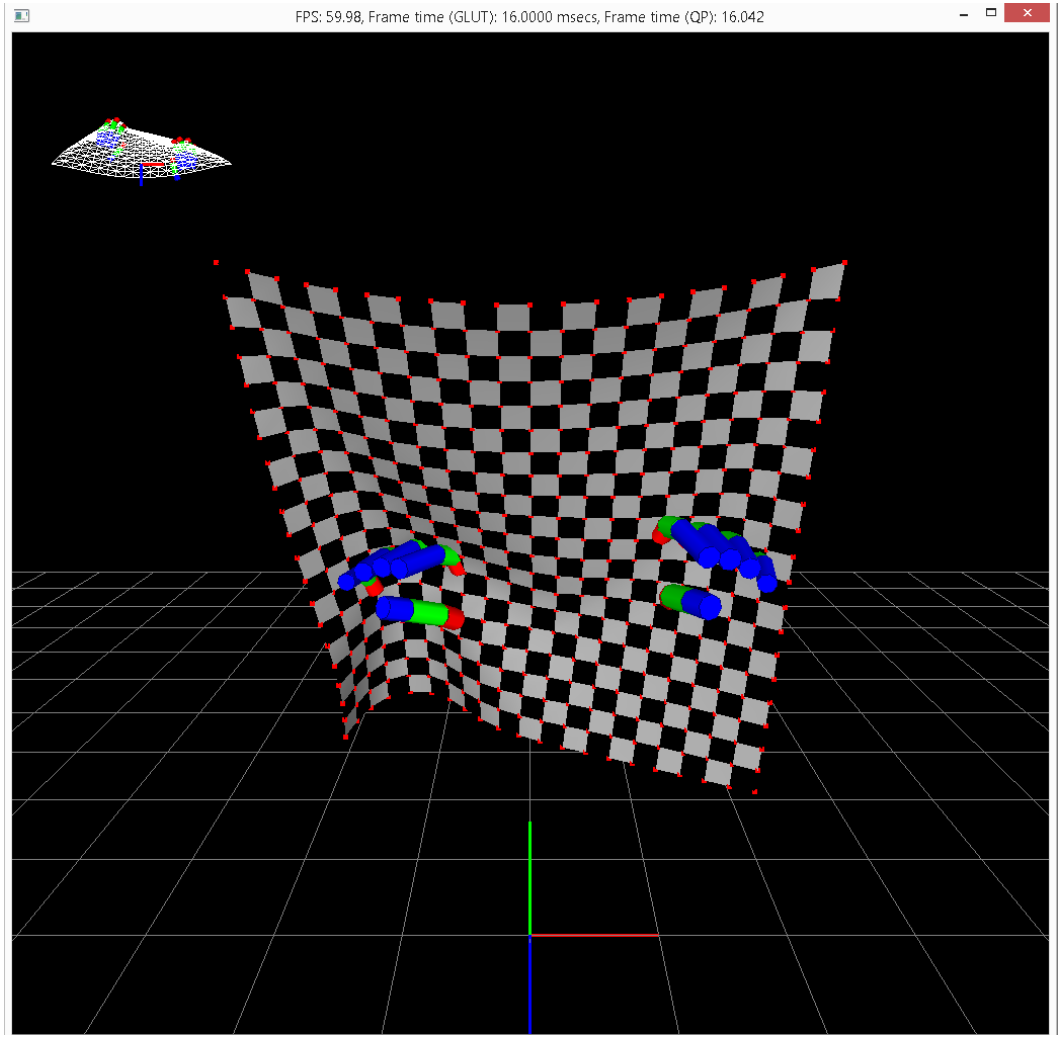
Photorealistic 3D environments

Physics Simulation



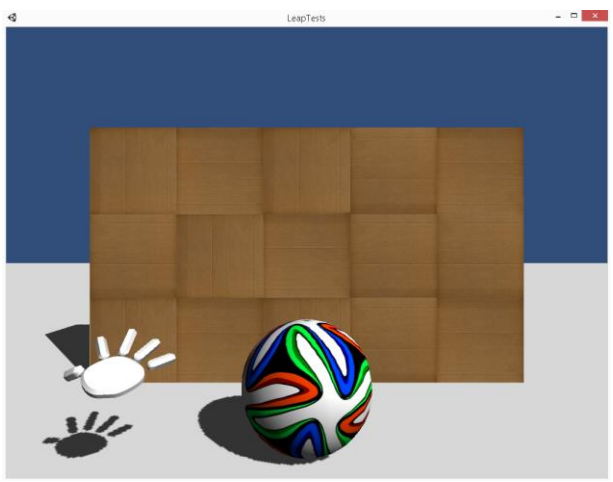
Fire Simulation Demo

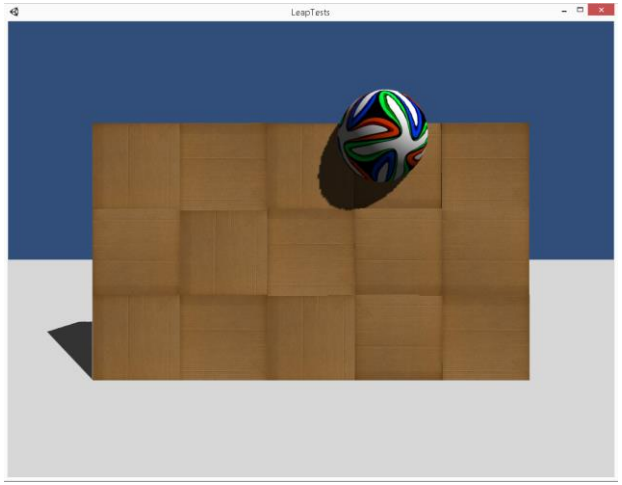
Physics Simulation

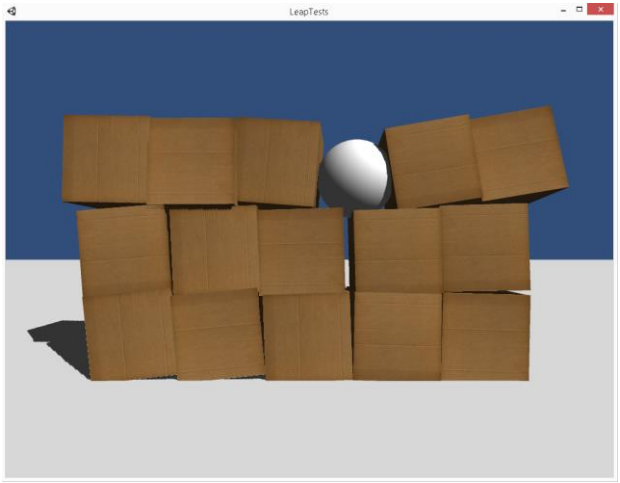


Interactive Cloth Simulation with Leap Motion

Physics Simulation

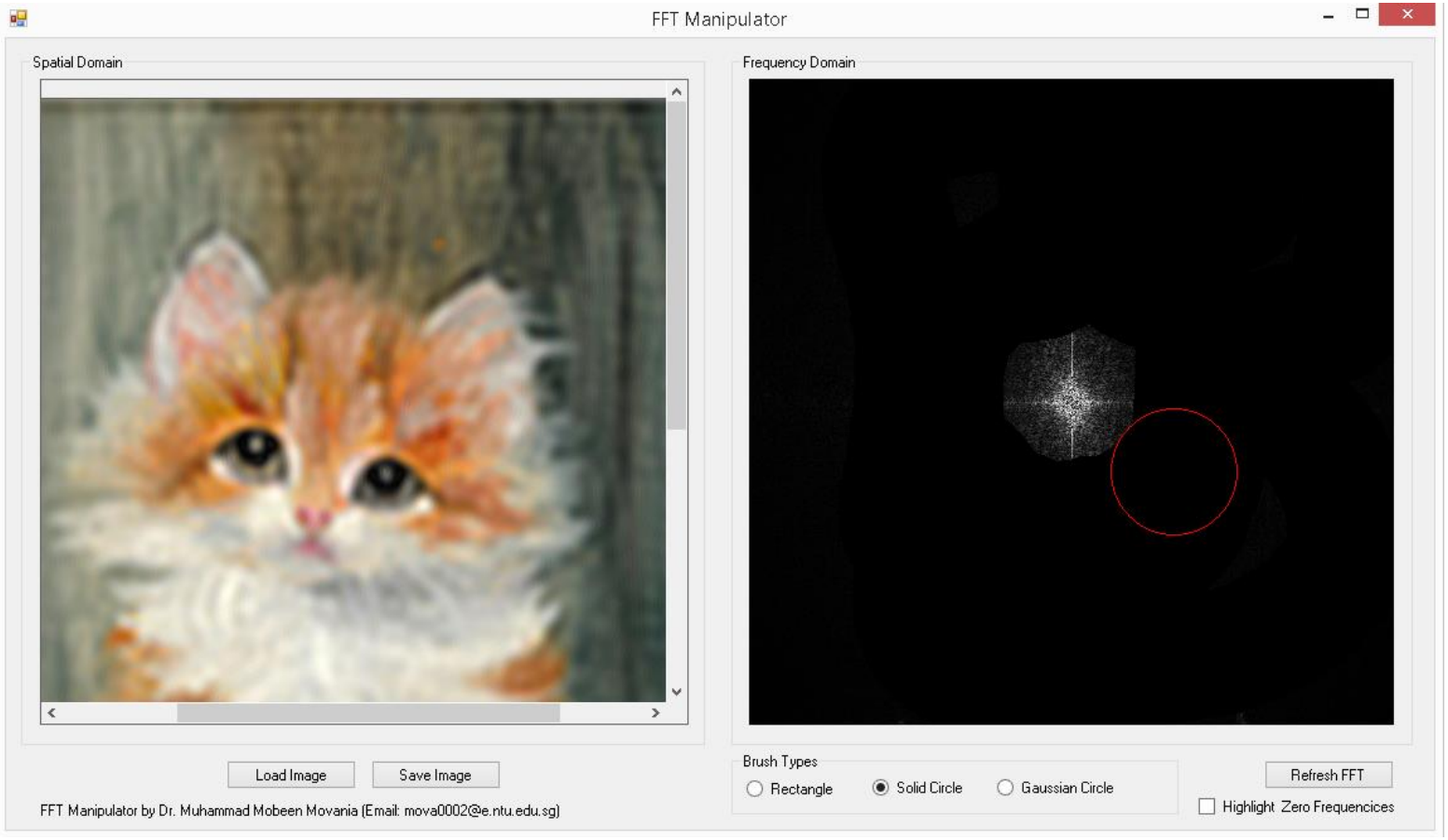






Grab and Throw - Leap Motion Demo

FFT Manipulator



FFT Manipulator Demo

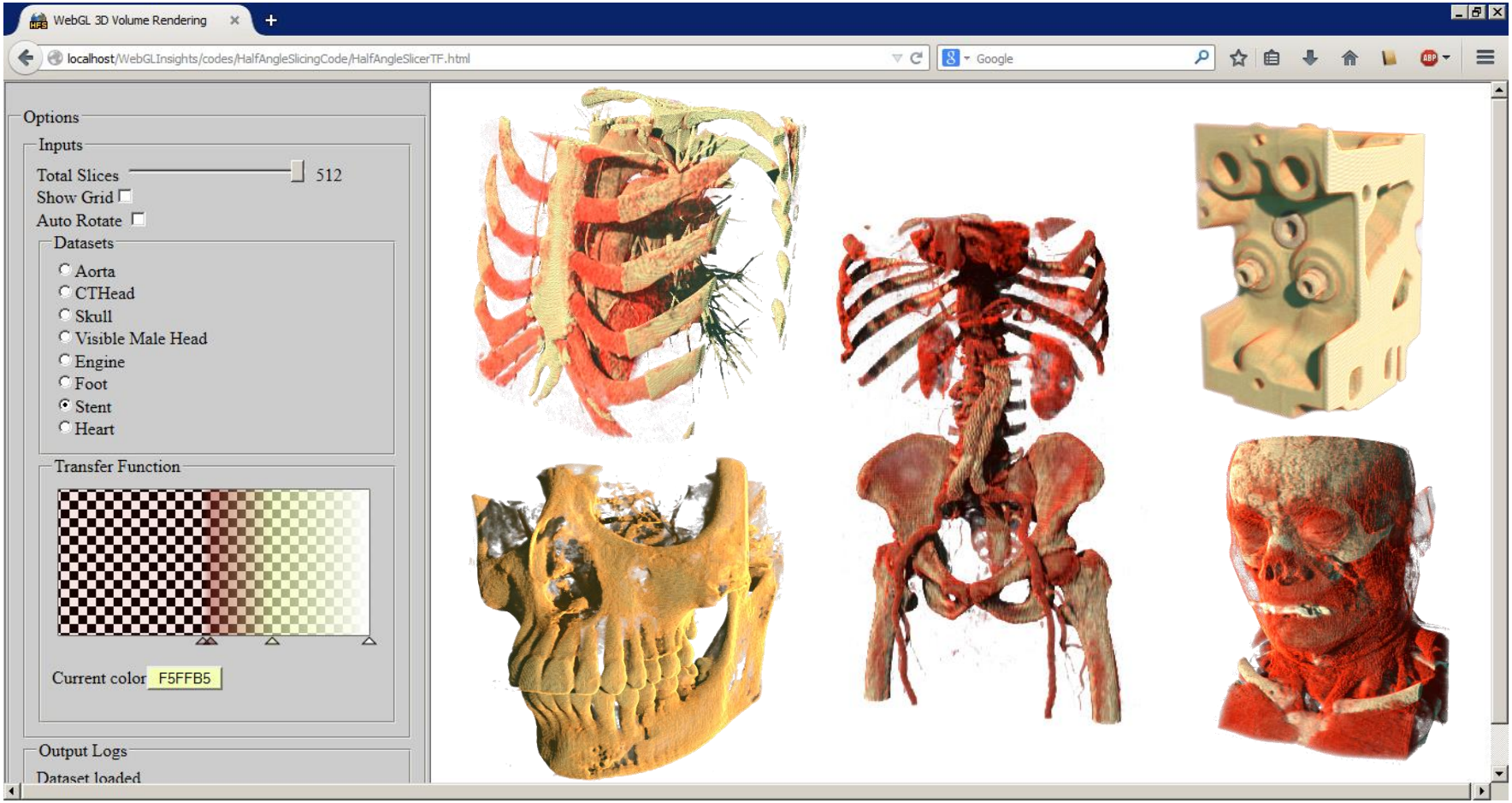
Fracture simulation

Vase Smasher Demo

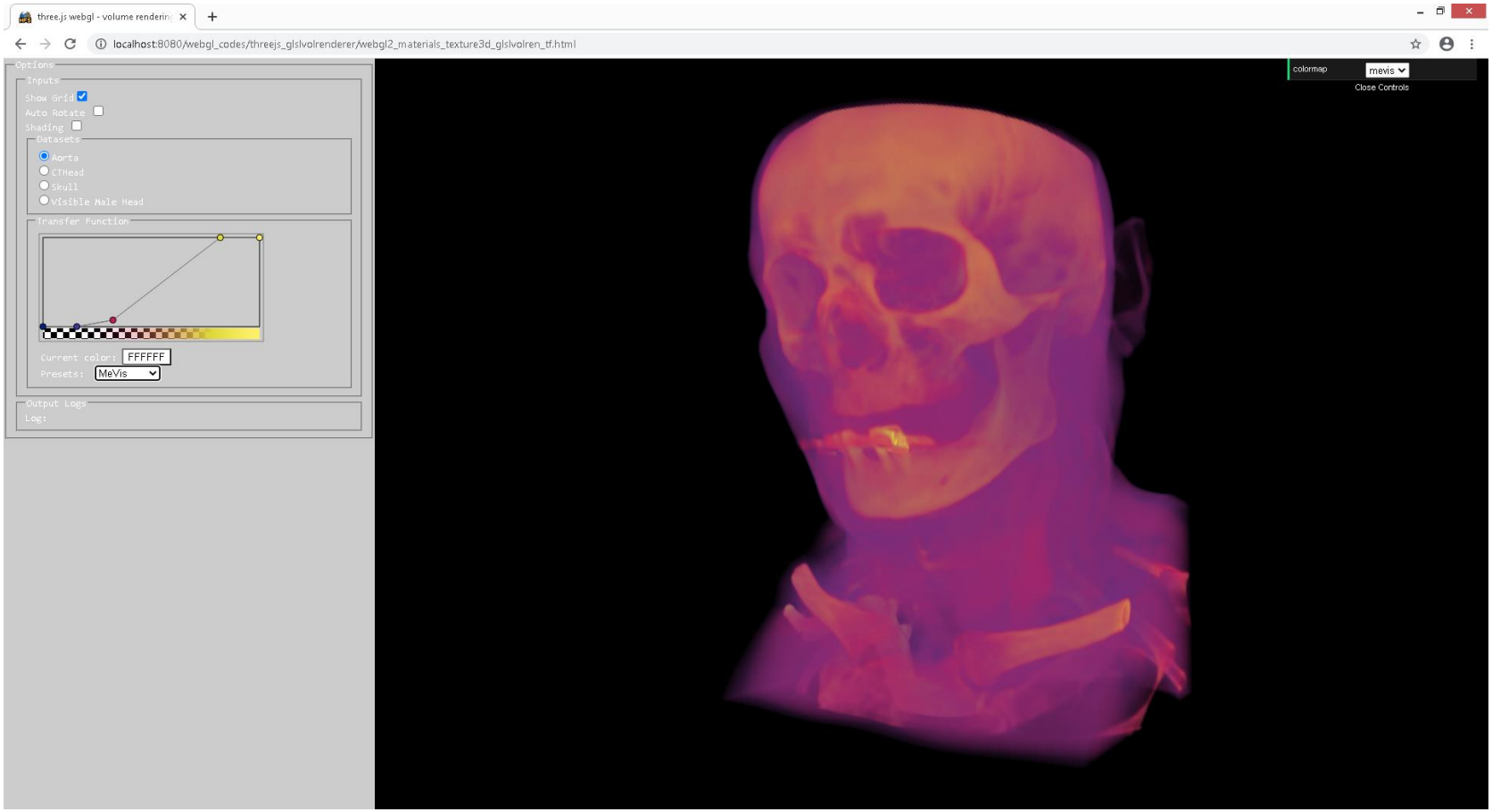
Translucent material rendering

Translucency Demo

WebGL Volume Rendering

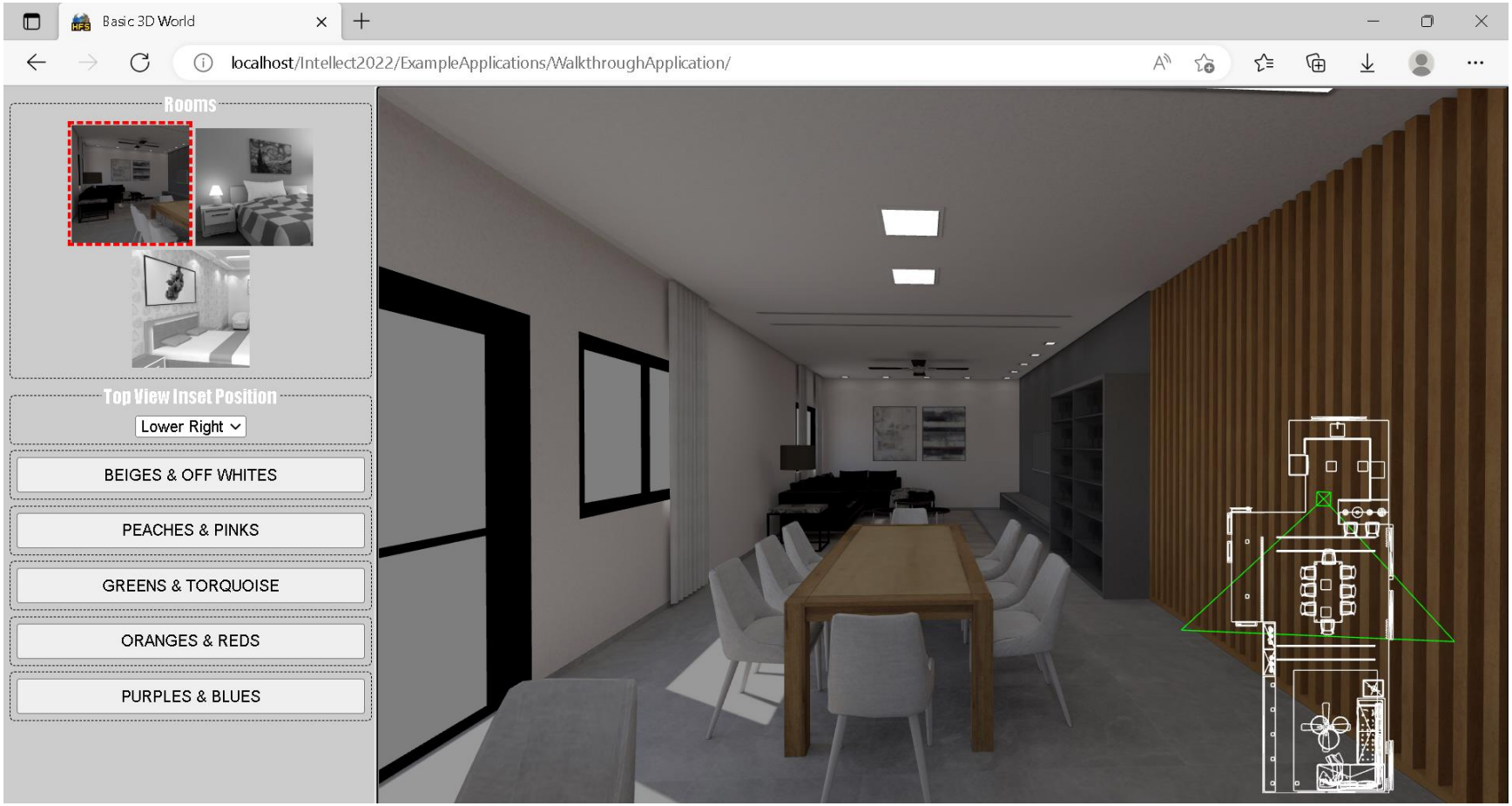


Browser based Volume Rendering



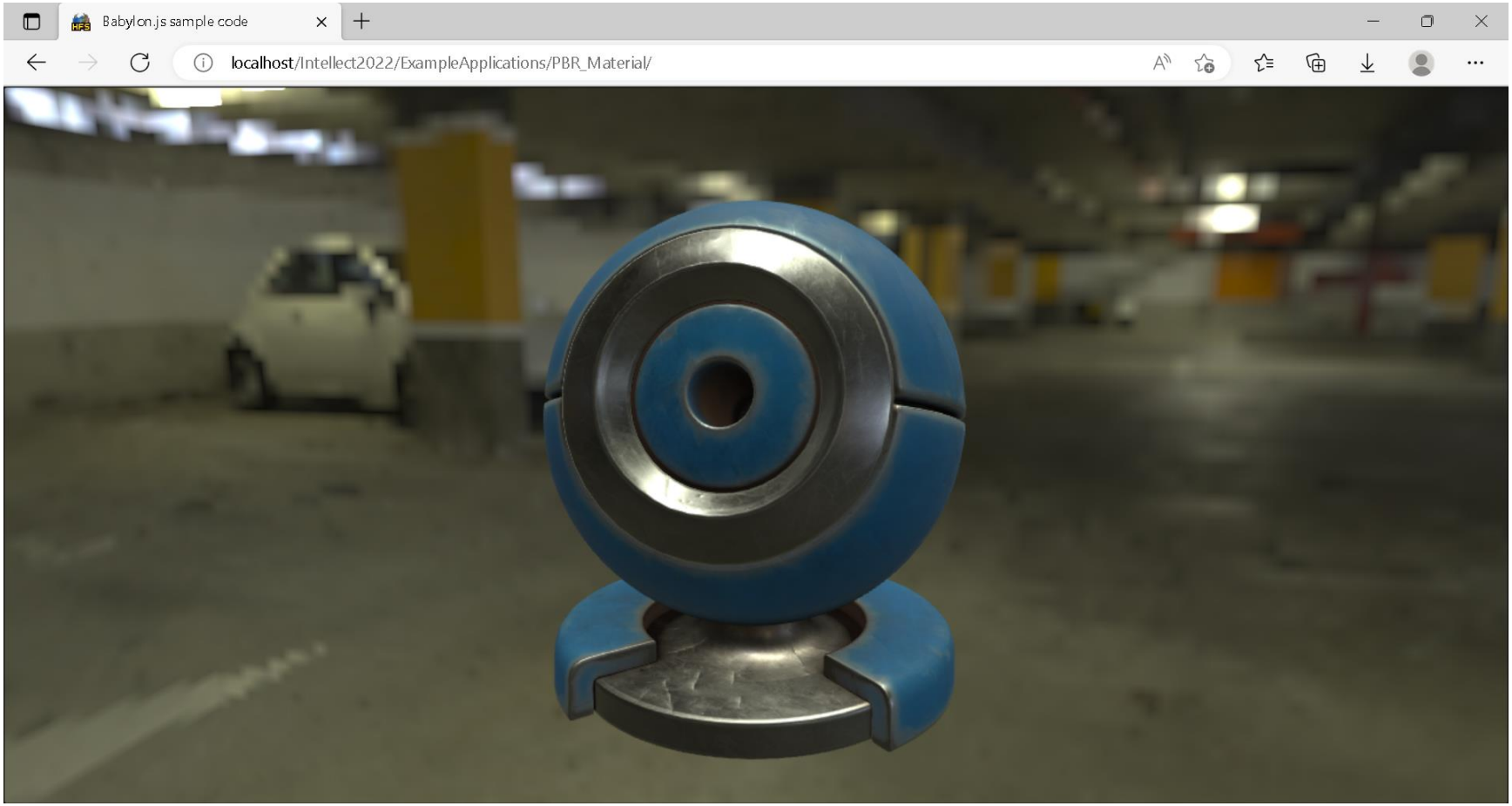
Browser based Volume Rendering Demo

Photorealistic Walkthrough



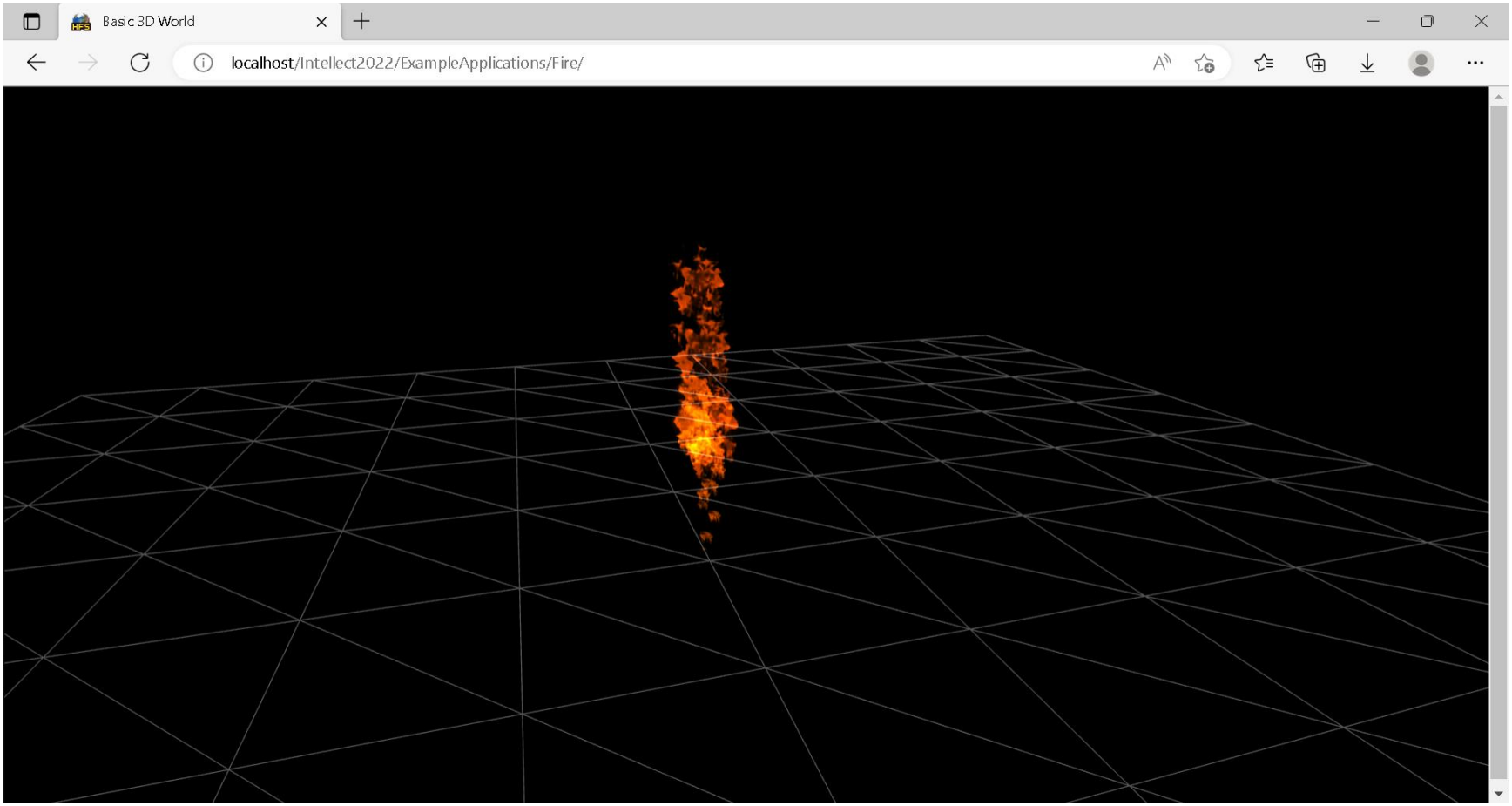
WalkThroughApplication

Physically based Materials



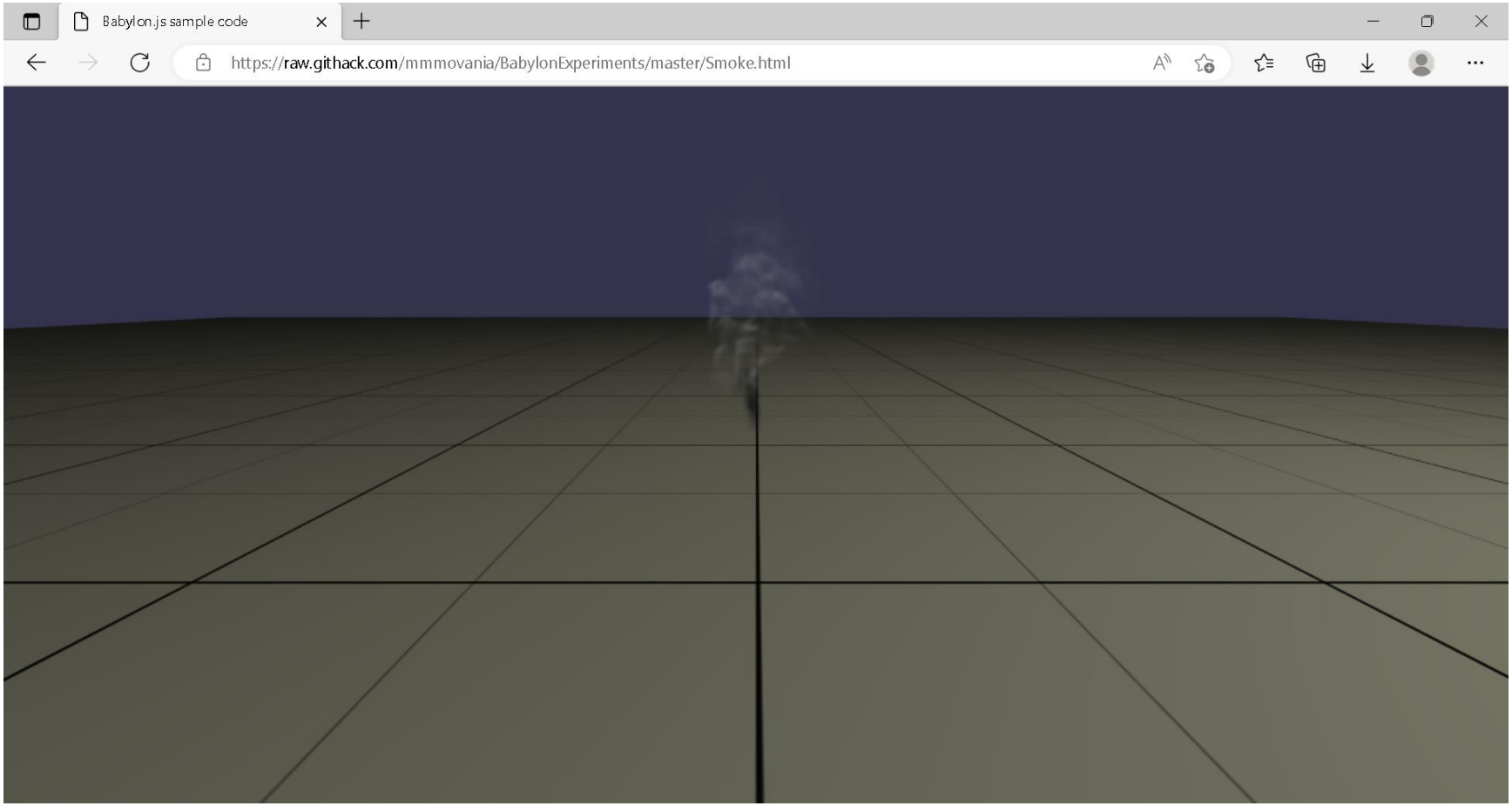
PBR_Material

Particle based Fire Simulation



Particle based Fire Simulation

Smoke Simulation



https://raw.githack.com/mmmovania/BabylonExperiments/master/Smoke.html

Physics Simulations

- Thanks to availability of good javascript physics engines like physi.js, ammo.js and cannon.js, its relatively easy to integrate them into our own WebGL applications.
- Separate set of tutorials for three.js and Babylon.js
  - https://github.com/mmmovania/Physijs_Tutorials
  - https://github.com/mmmovania/BabylonPhysicsTutorials

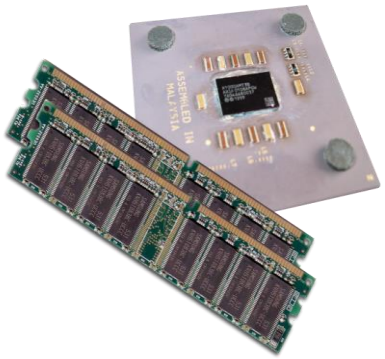
Conclusion

- Historically, working in computer graphics was tough because of non-availability of convenient API.
- With the introduction of WebGL and libraries like three.js and Babylon.js, we can do modern graphics programming in a web browser.
- Ease of use, great tools for code debugging and direct shader access allow rapid prototyping of ideas.

High Performance Computing using CUDA

Terminologies

- Heterogeneous computing
  - Computation involving two different kinds of processors:
    - Host (CPU)
    - Device (GPU)



Host (CPU)

Device (GPU)

What's a Kernel

- Kernel
  - A special function that is run on the device (GPU) but is called from the host (CPU)
  - Any data that is required for processing is passed as a parameter to the kernel function much like normal functions in C
  - Note that apart from simple variables, the parameters that are passed to the kernel as pointers must be in device memory (CUDA sdk provides special functions to copy data to from host to device memory and copy data from device to host)
  - Note that the kernel function call is asynchronous

What is Execution Configuration?

- Execution configuration
  - When a kernel function is called from the host (CPU), it must provide the number of blocks and the number of threads in each block
  - This prepares the number of threads on which the kernel function will be called
  - Usually, the execution configuration is enclosed in trippled angled brackets e.g. myKernel<<<32,32>>>();

The Big Picture

- As usual, the program execution starts at main (single thread) on CPU (host)
- When there is some GPU work to be done, the CPU (host) calls one or more GPU (device) functions (called kernels)
  - Kernel is the processing function which is called for the given number of threads (execution configuration)
  - Kernel functions callable from host are given a special qualifier __global__ in their signature
  - To make a function a device function so that it can be executed on the device (GPU), we must include __device__ qualifier in its signature

How to get started?

- You need to have an NVIDIA graphics hardware to run CUDA
- If you do not have NVIDIA hardware, you have two options:
  - Use Google Colab
  - Use Kaggle

Getting Started (If you have NVIDIA hardware)

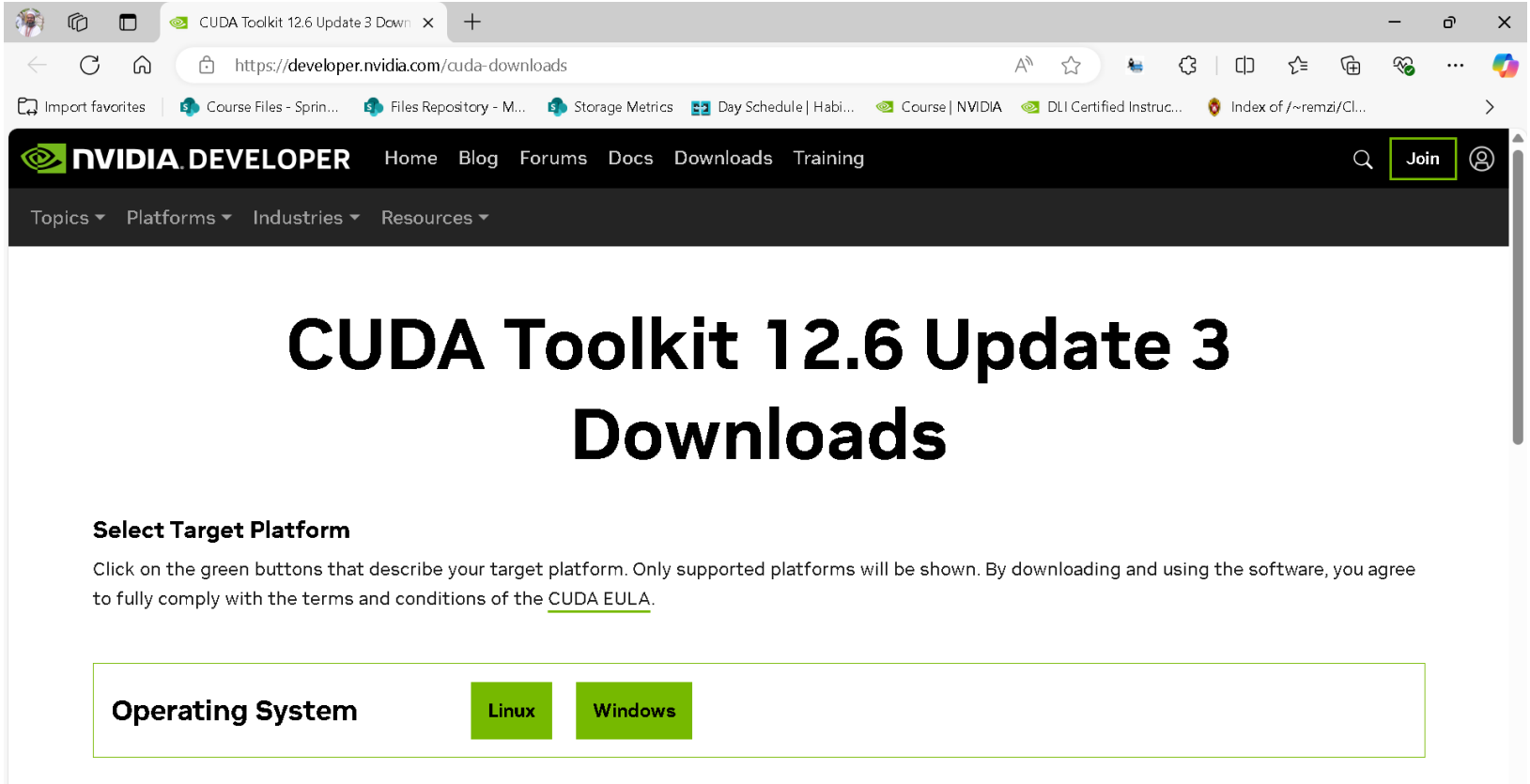
- Head over to NVIDIA website and download the latest CUDA GPU Computing SDK from https://developer.nvidia.com/cuda-downloads

- Download latest CUDA version.

- Ensure that you download the correct installer based on your platform

Getting Started
(If you have NVIDIA hardware)

- CUDA SDK download



Getting Started (If you have NVIDIA hardware)

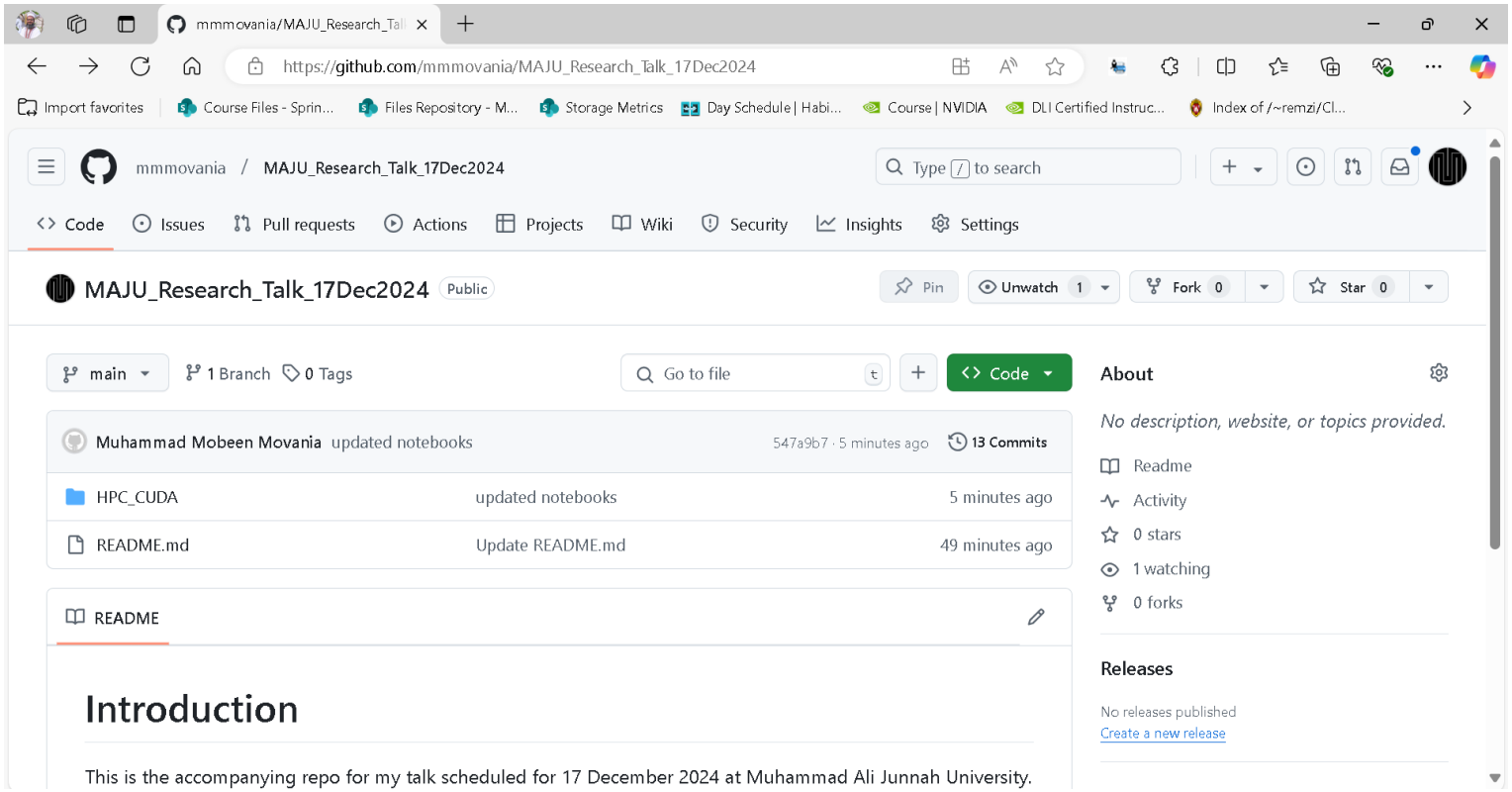
- Compilation on Windows
- Two options
  - Using command prompt
  - Using Visual Studio IDE

- Using command prompt
  - Assuming that CUDA is installed correctly and nvcc is in PATH
  - Open the Visual Studio command prompt
  - Assuming that the source file is named Hello.cu, change directory to the folder containing the Hello.cu file
  - Next invoke nvcc like this and it will generate the Hello.exe file
    - nvcc Hello3.cu –o Hello.exe

- Using Visual Studio IDE
  - After installation open the CUDA samples and build them using your Visual Studio IDE
  - You may create a new Win32 sample and add the CUDA sources (shown later)

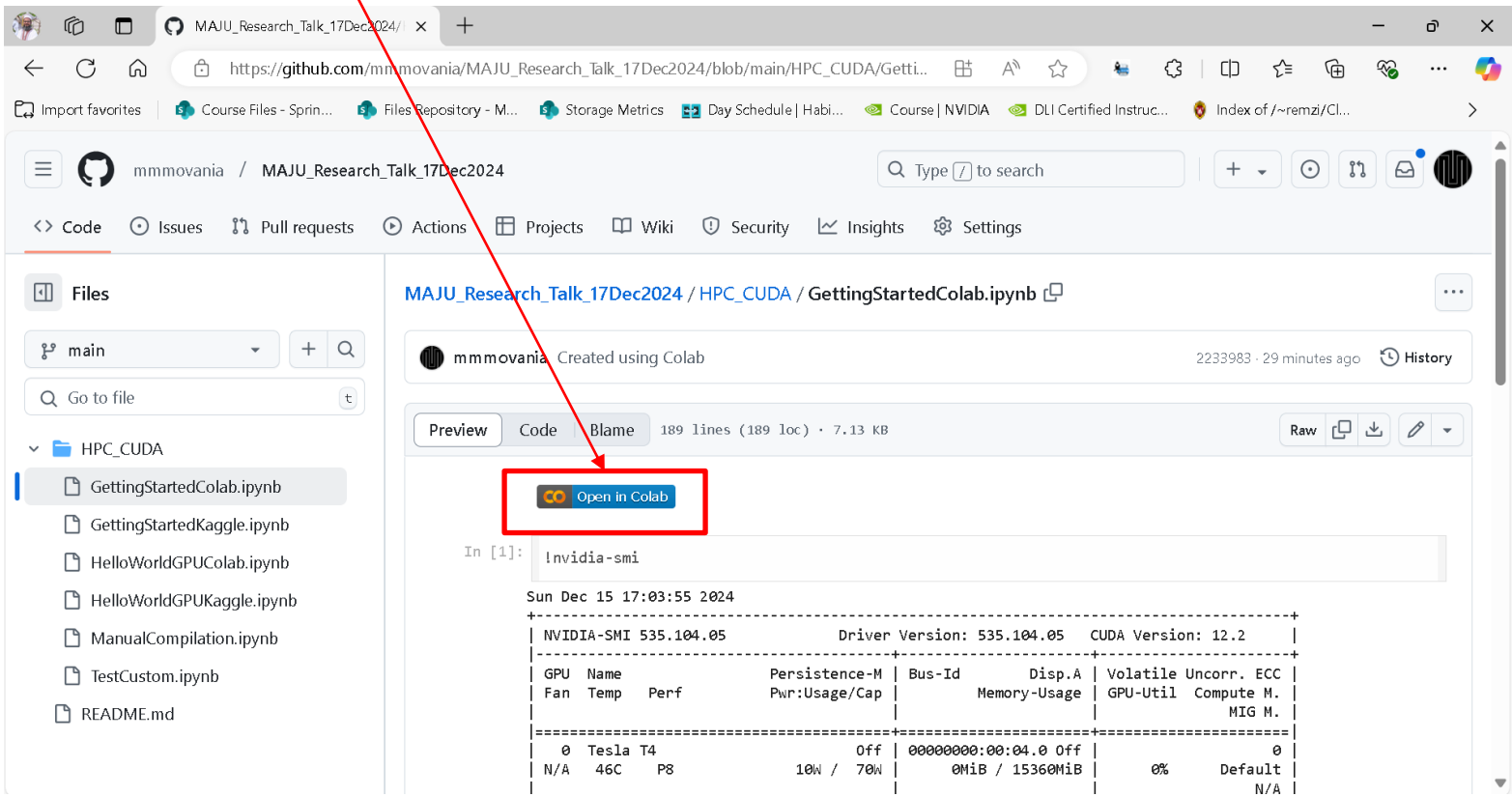
Getting started
(If you do not have NVIDIA hardware)

- A [github repo]() with basic setup steps for using CUDA C on Google colab or Kaggle is maintained.



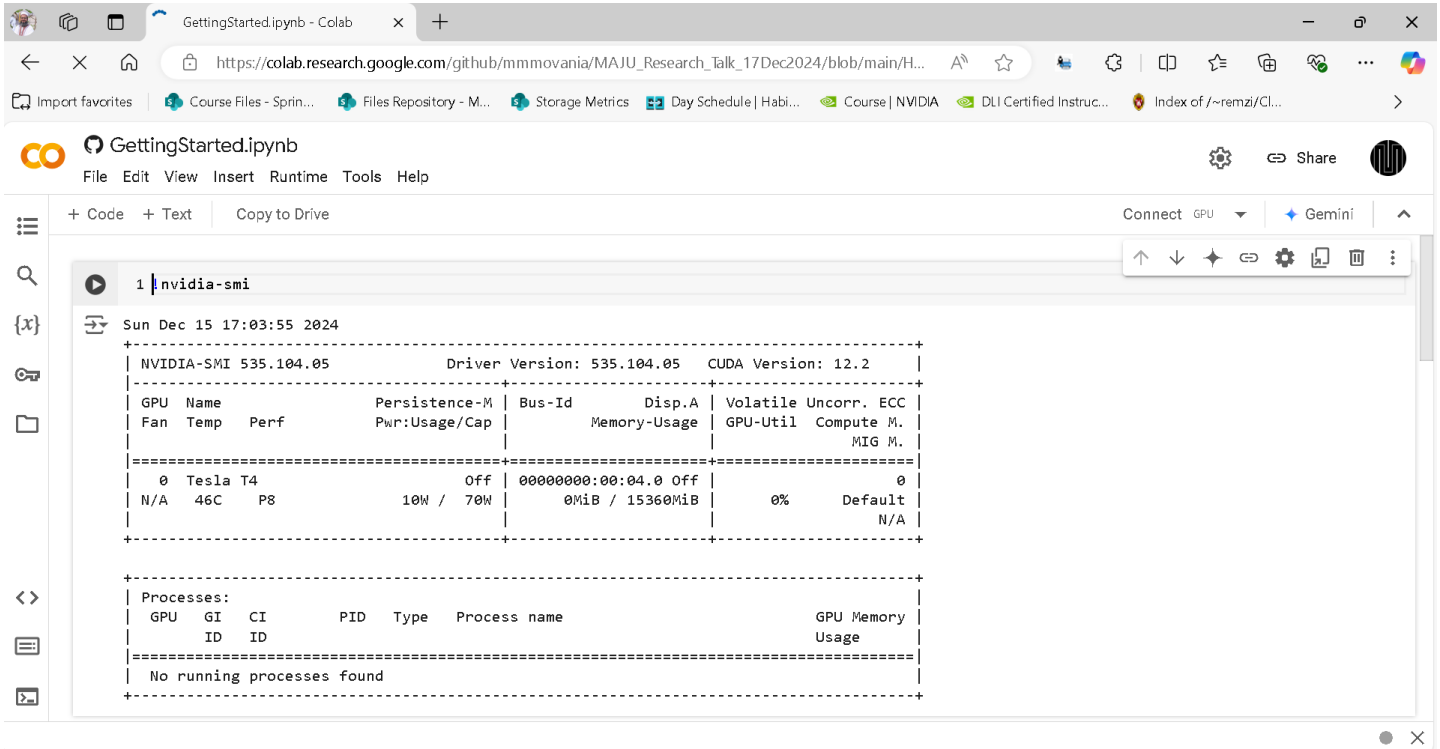
Getting started
(If you do not have NVIDIA hardware)

- Go to HPC_CUDA and open GettingStarted.ipynb
- click open in colab button



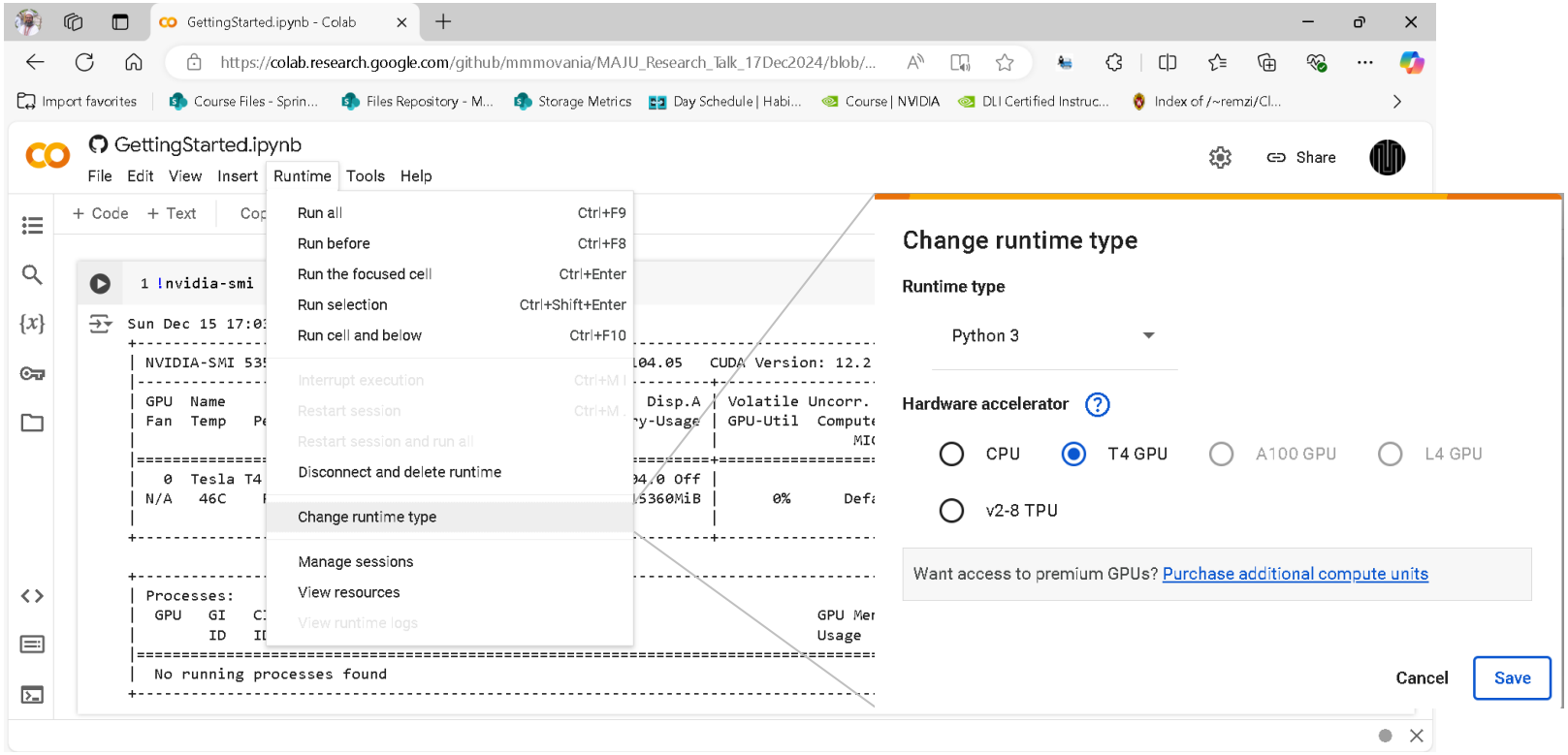
Getting started
(If you do not have NVIDIA hardware)

- You will be presented with this screen, go to File and Save a Copy in Drive to create your own copy of code



Getting started
(If you do not have NVIDIA hardware)

- Go to Runtime->Change runtime type and then select GPU from the combobox and press Save



Basic Code Examples

Hello World on host (Hello1.cu)

```
int main(void) {
    printf("Hello World!\n");
    return 0;
}
```

- Standard C that runs on the host
- NVIDIA compiler (nvcc) can be used to compile programs with no *device* code

Output:

```
$ nvcc Hello1.cu
$ a.out
Hello World!
$
```

Hello World! with Device Code (Hello2.cu)

```
__global__ void mykernel(void) {
}

int main(void) {
    mykernel<<<1,1>>>();
    printf("Hello World!\n");
    return 0;
}
```

Output:

```
$ nvcc Hello2.cu
$ a.out
Hello World!
$
```

- Two new syntactic elements…

Hello World! with Device Code

```
__global__ void mykernel(void) {
}
```

- CUDA C/C++ keyword `__global__` indicates a function that:
  - Runs on the device
  - Is called from host code

- `nvcc` separates source code into host and device components
  - Device functions (e.g. `mykernel()`) processed by NVIDIA compiler
  - Host functions (e.g. `main()`) processed by standard host compiler
    - `gcc, cl.exe`

Hello World! with Device COde

```
mykernel<<<1,1>>>();
```

- Triple angle brackets mark a call from *host* code to *device* code
  - Also called a “kernel launch”
  - We’ll return to the parameters (1,1) in a moment

- That’s all that is required to execute a function on the GPU!

Hello World! with Device Code

```
__global__ void mykernel(void){
}

int main(void) {
    mykernel<<<1,1>>>();
    printf("Hello World!\n");
    return 0;
}
```

- `mykernel()` does nothing, somewhat anticlimactic!

Output:

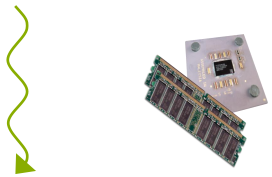
```
$ nvcc Hello2.cu
$ a.out
Hello World!
$
```

Hello World with Device Code (Hello3.cu)

```
#include <stdio.h>

__global__ void HelloKernel() {
   printf("\tHello from GPU
          (device)\n");
}

int main() {
   printf("Hello from CPU (host) before
           kernel execution\n");

   HelloKernel<<<1,32>>>();

   printf("Hello from CPU (host) after
           kernel execution\n");

   return 0;
}
```

parallel function (kernel)

serial code



parallel code

serial code

Output

Developer Command Prompt for VS2012

```
   Creating library Hello.lib and object Hello.exp


Hello from CPU (host) before kernel execution
Hello from CPU (host) after kernel execution
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
```

Things to be noted

- The CPU code executes first and then invokes GPU code asynchronously
- The CPU execution continues to the next statement immediately after the kernel launch
- What if we want the CPU execution to wait for the GPU to finish and then continue its execution
  - Call **cudaDeviceSynchronize** function after kernel launch
  - This functions forces CPU to wait for the GPU execution to finish

Hello World Synchronized (HelloSynched.cu)

```
#include <stdio.h>

__global__ void HelloKernel() {
    printf("\tHello from GPU (device)\n");
}

int main() {
    printf("Hello from CPU (host) before
            kernel execution\n");

    HelloKernel<<<1,32>>>();

    cudaDeviceSynchronize();

    printf("Hello from CPU (host) after
            kernel execution\n");
    return 0;
}
```

Output

Developer Command Prompt for VS2012

```
Hello from CPU (host) before kernel execution
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
        Hello from GPU (device)
Hello from CPU (host) after kernel execution
```

Thanks

Any questions?

Please feel free to get in touch should you need any more information

Email: mobeen.movania@sse.habib.edu.pk

mova0002@e.ntu.edu.sg