Progetti per il Corso di Computer Graphics

Modalità di Svolgimento

- Scegliere uno dei progetti elencati sotto. La scelta dell'argormento è lasciata allo studente e non ci sono limitazioni alecune.
- I progetti sono da svolgersi singolarmente o in gruppo. Notate che per ogni progetto è indicato il numero massimo di partecipanti. Non è possibile svolgere un progetto con più partecipanti del numero massimo. Per quanto sia possibile svolgere il progetto con meno partecipanti, quella modalità è fortemente sconsigliata.
- Il voto del progetto sarà assegnato in base alla qualità dei risultati e alla loro presentazione (immagini e PDF). Non si prenderà in considerazione la "qualità" del codice, nè i tempi di esecuzione.
- È possibile consultare e utlizzare codice trovato online. In questo caso, **il codice usato va citato** indicandolo nel PDF in modo visibile. Notate però le limitazioni di cui sotto sull'uso del codice.

Modalità di Consegna

- La consegna del progetto deve contenere
    - codice scritto
    - immagini che dimostrano l'algoritmo
    - dati per eseguire i tests
    - un PDF che contiene: (1) nome degli studenti, (2) breve descrizione di quello che si è fatto, (3) risultati commentati
- Il progetto va consegnato su Classroom, **in un unico zip**. Ogni membro del gruppo consegna lo stesso file.
- Consegnare il progetto almeno 5 giorni prima della data dell'appello, per permettere di formulare un voto.
- Il voto sarà comunicati attraverso Classroom e convalidato dopo l'orale.

Materiale

- I progetti possono essere svolti il C++ o Python. Per ogni progetto, indicherò la fattibilità di ogni lavoro nel linguaggio specifico. Quest'ultima indicazione è da considerarsi un consiglio. Lo studente è libero di eseguire il progetto come meglio crede.
- Per i progetti in C++, consiglio di usare la libreria Yocto/GL, usando il setup fatto per gli esempi in classe. Basta scaricare il codice fatto in classe e aggiungere un esempio in più. La libreria è disponibile anche da Yocto/GL dove ne trovate la documentazione.
- Librerie utili in C++ per manipolazione di immagini:
  - CImg,
  - Piccante.
- Librerie necessarie in Python:
  - NumPy,
  - Scipy.
- Librerie utili in Python per manipolazione di immagini:
  - Pillow,
  - OpenCV.
- Librerie utili in Python per manipolazione di mesh:
  - TriMesh.
- Il docente può rispondere a domande sulla libreria Yocto/GL ma non le altre.

Progetti: Immagini

- **Exposure Fusion per Effetti HDR**: 2 persone in C++ o Python
  - A lezione abbiamo visto come le immagini HDR si ricoscono da una sequenza di immagini LDR. Per poi prendere la immagini HDR e tornare a LDR per il display.
  - Una alterantiva è quella di fondere diverse immagini LDR direttamente in una immagine LDR, che meglio rappresenti i colori delle immagini originali.
  - Implementare Exposure Fusion
  - Variazioni di questo algoritmo sono usate in vari telefoni.
- **Controllo Locale del Contrasto**: 2 persone in C++ o Python
  - Abbiamo visto a lezione come controllare il contrasto di una immagine in modo globale
  - In questo progetto faremo invece un controllo locale del contrasti
  - Implementare Local Laplacian Filters

    - Variazioni di questo algoritmo sono usate in vari programmi di image processing.
- **Image Cloning**: 3 persone in C++ o Python
    - Vari programmi permettono di fare blending di immagini senza vedere i bordi.
    - In questo progetto implementiamo Coordinates for Image Cloning
- **Stippling**: 2 persone in C++ o Python
    - implementare Wieghted Voronoi Stippling
    - possibile implementazione da sequige da qui
- **Stippling2**: 3 persone in C++ o Python
    - implementare Weighted Linde–Buzo–Gray Stippling
    - varie possibili implementazioni su Github da cui ispirarsi
- **Cloning**: 3 persone in C++
    - implementare PatchMatch
    - varie possibili implementazioni su Github

Progetti: Rendering

- **Rendering Volumetrico**: 1 persona in C++ (max 24 punti)
    - Implementare il rendering di rifrazione e volummi come presentato in Ray Tracing in One Weekend
- **Triplanar Mapping**: 1 persona in C++ (max 24 punti)
    - Implementare triplanar mapping come spiegato in articoli onlines come 1 o 2
    - L'implementazione deve modificare il renderer cambiando `eval_texture` in modo appropriato e memorizzando le informazioni necessarie in `texture_data`
- **Raycasting di Superfici Implicite**: 2 persone in C++
    - Implementare un renderer che utilizza superfici implicite come descritto il classe.
    - Definire una nuova scena, dove ogni shape è un modello implicito.
    - I materiali si possono definire come colori nella shape o come uv dalla shape (la prima è la più facile)
    - Ogno shape sarà un parallelepipedo che quindi può essere incluso in un bvh. Ma per questo progetto va anche bene fare intersezioni non accelerate.
    - Usare nanovdb per rappresentare dati volumetrici
- **Mip-mapping**: 2 persone in C++
    - Modificare il renderer per supportare mipmaps
    - Aggiungere le piramidi di immagini a ogni texture usando `image_resize()`

    - Aggiungere ray differentials per determinare i livello delle texture da cui fare il sampling; per questo progetto basta fare mipmapping solo dalla telecamera.
    - Usare come ispirazione l'implementazione da Pbrt
- **Texture Synthesis**: 2 persone in C++
    - Implementare un generatore di textures da esempi seguendo l'algoritmo da On Histogram-Preserving Blending for Randomized Texture Tiling
    - L'implementazione di può fare aggiungendo i dati necessari all'oggetto `texture_data`
- **Other BVHs**: 1 persona in C++
    - il tempo di esecuzione del raytracer è dominato da ray-scene intersection; testiamo altri BVH per vedere se ci sono speed up significativi
    - abbiamo già integrato Intel Embree, quindi questo non conta
    - integrare il BVh nanort e bvh
    - testare le scene più larghe e riportare nel readme i tempi confrontandoli con i nostri
    - per integrarli, potete scrivere nuovi shaders o inserire i nuovi Bvh dentro un BVH generico come fatto in `trace_bvh`, ma aggiungendolo al raytfacer fatto in classe

Progetti: Modellazione

- **Foreste**: 1 persona in C++
    - Creare scene molto complesse usando gli assets da PolyHaven
    - Per ogni asset, piazzare l'asset su una superficie con probabillità che dipende da una texture, usando varie textures per controllare oggetti diversi
    - Scene come foreste probabilmente funzionano bene. Ma potete, e anzi dovreste, utilizzare altri oggetti se possibile.
- **Sample Elimination**: 1 persona in C++
    - scrivere una funzione che posiziona punti su una superficie arbitraris; mostrare i risulati mettendo punti o piccole sfere sulla superficie e confrontando random sampling e questi metodi
    - seguire la pubblicazione sample elimination
    - l'algoritmo richiede di trovare dei nearest neighbors; per farlo usate o `hash_grid` in Yocto/Shape o `nanoflann` su GitHub
- **Poisson Point Set**: 1 persona in C++
    - seguire la pubbliacazione in poisson sampling
    - trovate varie implementazioni di questo metodo, inclusa una dell'autore
    - dimostrate il risultato mettendo punti su un piano

- **Libreria di Noise**: 1 persona in C++
  - Implementare una funzione generica che produce vari tipi di noise
  - Come base per il noise, usare la funzione data in classe per gradient noise
  - Implementare variazioni di noise come
    - value noise
    - gradient noise
    - voronoi e smooth voronoi
    - voronoise
    - voronoise edges
    - variazioni frattali di tutti i noise
  - Dimostrare i vari tipi di noise in 2D e 3D
- **Alberi I**: 2 persone in C++ o Python
  - implementare un generatore di alberi che segue l'algoritmo presentato qui
  - esempio con animazioni
  - esempio di risultati artistici
  - per ogni segnmento potete usare un cono troncato o una capsula (cono con sfere alla fine)
  - Per le foglie usate forme stilizzate, come poligoni piatti che approssimano le foglie
- **Alberi II**: 3 persone in C++ o Python
  - come alberi I, ma generando anche le textures
  - per le foglie, mettere textures con alpha matting su quad; trovate varie textures di foglies online
  - per l'labero potete implementare Triplanar Mapping nel nostro renderer, o provare a generare delle texture coordinates che rispettino le dimensione dei rami e usare texture2d prese dal web