

Voxel Renderer

I. Scopo

Sviluppare un'app Android per la visualizzazione interattiva di mesh discretizzate tramite voxels.

Un voxel (Volumetric Picture Element) rappresenta un dato posizionato all'interno di una griglia discretizzata in tre dimensioni. Tale griglia, chiamata anche occupancy grid, è quindi indicizzabile tramite una terna di interi e ciascun elemento di una griglia può o non può essere occupato da un voxel. In caso positivo di occupazione, il voxel è caratterizzato da una struttura dati arbitraria. Per esempio, una terna RGB indicante il suo colore.

Come accade per le mesh viste a lezione, esistono diversi formati in grado di rappresentare una occupancy grid di un volume discretizzato a voxel. Per questo progetto, si consideri il formato descritto nella sezione II di questo documento. L'app Android da sviluppare dovrà quindi essere in grado di interpretare tale formato, e produrne una visualizzazione su cui l'utente potrà interagire.

Dato il potenziale alto numero di voxel sullo schermo, si chiede allo studente di ottimizzare il più possibile il numero di drawcall e l'ammontare di dati passato da host a device.

II. Specifiche del formato

Il formato in questione è stato inventato appositamente per questo progetto. Si chiama "vly", in quanto somiglia allo *Stanford ply* visto a lezione, ma adattato per le occupancy grids. E' un formato testuale di cui si riporta un esempio triviale:

```
Filename: simple.vly
grid_size: 1 1 3
voxel_num: 3
0 0 0 0
0 0 1 1
0 0 2 2
0 238 0 0
1 0 238 0
2 0 0 238
```

Come si può notare, il formato vly ha due righe di preambolo. Nella prima, si evince la dimensionalità massima dell'occupancy grid nelle tre dimensioni. Nella seconda, il numero di

elementi della griglia effettivamente occupati da un voxel. In questo esempio, il numero totale di elementi della griglia coincide con il numero totale di voxel esistenti, ma questo non sarà sempre vero.

Dopo il preambolo, le successive *voxel_num* righe di testo sono nella forma:

<Xi, Yi, Zi, Ci> con <Xi, Yi, Zi> è la terna di numeri interi positivi che esprime le coordinate di un voxel all'interno dell'occupancy grid. Ci è l'indice del colore, anch'esso un intero.

Tale intero è un riferimento alla successiva parte del file, quella che comincia dopo il preambolo e dopo la descrizione di ciascun voxel.

La seconda parte è nella forma <Ci, R, G, B>, con Ci indice del colore e successivamente una terna RGB indicante il colore.

In poche parole, la prima parte del file descrive dove sono i voxel e ne indica un colore tramite un intero che funge da indice di riferimento. La seconda parte del vly, è un set di colori, ciascuno identificato da una chiave intera. Implementando un render di quel file:



Si noti come il formato in questione consideri la Z come up-axis, e che ciascun canale di colori vada da 0 a 255.

III. Specifiche di implementazione

La distanza tra l'origine della camera e l'oggetto voxellizzato deve, al netto di eventuali input dell'utente (si veda la sezione successiva), garantire una visuale il più possibile completa e coerente indipendentemente dal numero di voxel disegnati.

Una **soluzione banale** per il rendering dell'oggetto è quella di costruire un parallelepipedo per ogni voxel. Si dovrà quindi "tradurre" le coordinate dell'occupancy grid in coordinate spaziali del baricentro di ciascun parallelepipedo, e poi per ciascuno di essi impostare le giuste trasformazioni e associare a ciascun vertice di ogni cubo il colore appropriato.

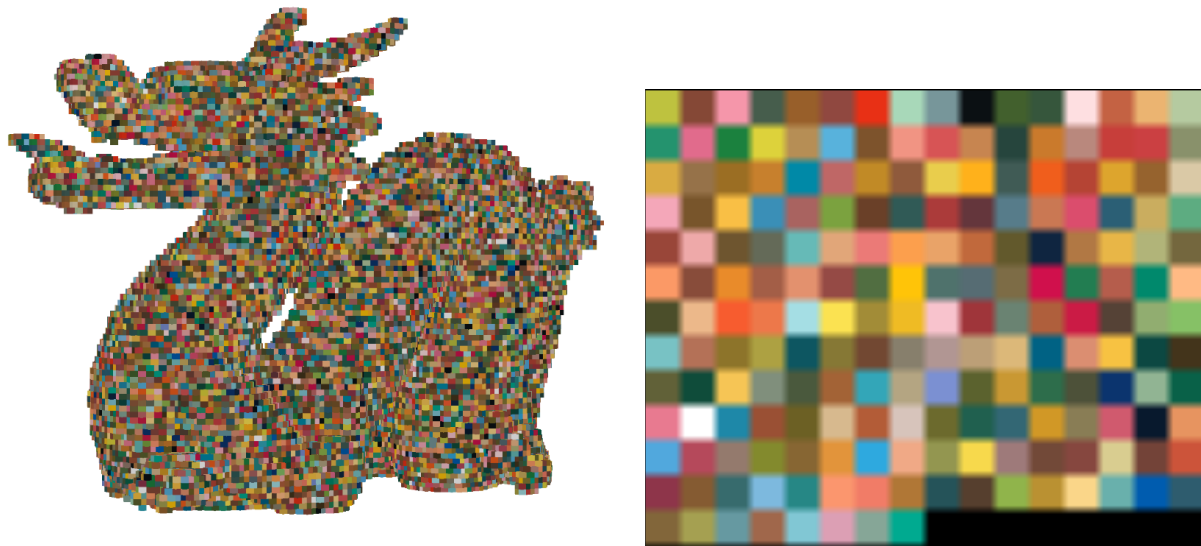
Dato l'alto numero di voxel, e quindi di parallelepipedi da disegnare, l'ammontare di drawcall e di dati da passare host-to-device graverebbe eccessivamente sulle performance dell'app.

Per migliorare questo aspetto esistono diverse soluzioni. Si invita lo studente ad implementare una soluzione che vada oltre quella appena descritta. Questo per ambire ad un voto più alto.

Si può pensare di “accorpare” voxel con colori simili in mesh uniche. O interi cluster di voxel generando proceduralmente le rispettive mesh.

In alternativa, il metodo migliore per minimizzare il numero di drawcall indipendentemente dal numero di voxel da trattare è rappresentato da *instanced rendering*. [Qui](#) una sua breve introduzione.

Per quanto riguarda la gestione dei colori, essa può essere ottimizzata generando al volo un'immagine (palette) da utilizzare come texture, generando quindi per i vertici di ciascun parallelepipedo delle texture coordinates appropriate per l'immagine generata. Per esempio:



L'immagine di destra funge da mappa dei colori per la mesh voxellizzata di sinistra. Questo implica la creazione di una [Android Bitmap](#), durante il parsing del vly. Tale bitmap verrà usata successivamente come buffer di dati per la creazione di un oggetto OpenGL texture.




IV. Specifiche di interazione con l'utente

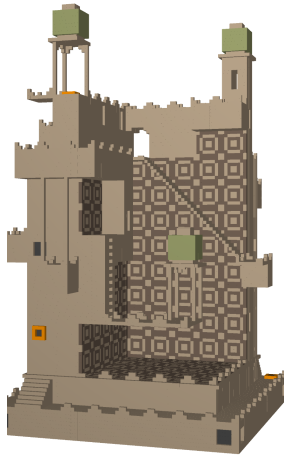
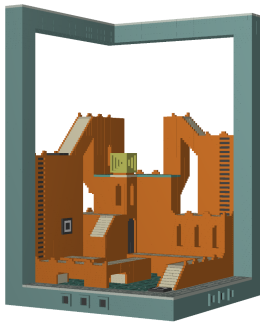

L'utente interagisce con la visualizzazione ruotando la mesh lungo l'asse delle altezze. In particolare, con eventi di tipo touch sulla parte destra e sinistra dello schermo per veicolare la direzione di rotazione.

Inoltre, tramite pinching (Scale listener), l'utente potrà eseguire uno zoom. Si vincoli lo zoom ad assumere valori consoni rispetto alla proiezione della mesh.

IV. Ulteriori dettagli

I file .vly da visualizzare sono descritti nella seguente tabella:

Nome file	Numero voxel	Immagine	Opera originale
simple.vly	3		-
chrk.vly	398		https://github.com/ephtarcy/voxel-model/tree/master
dragon.vly	40265		https://github.com/ephtarcy/voxel-model/tree/master

monu2.vly	150764		https://github.com/ephtarcy/voxel-model/tree/master
monu16.vly	194300		https://github.com/ephtarcy/voxel-model/tree/master
christmas.vly	230503		https://github.com/klucek/vox-models

La soluzione di base, deve essere almeno in grado di visualizzare le mesh prime due mesh di questa tabella.

Per il rendering, non si impone lo shading da utilizzare. Tutto ciò che non è stato esplicitamente dettagliato in questa specifica è soggetto all'intuizione ed interpretazione dello studente, fermo restando l'obiettivo di restituire una visualizzazione chiara per l'utente.

Elenco degli asset forniti

[Link](#)

- Modelli 3D
 - pcube.ply ASCII .ply format di un cubo [x,y,z,nx,ny,nz]

In spazio locale, questa mesh è centrata rispetto all'origine degli assi.

Tutti i file .vly elencati nella tabella della sezione IV.

V. Considerazioni di carattere generale

Come viene valutata l'app?

- Rispetto dei vincoli e descritti nelle sezioni I, II e III.
- Impegno percepito nello sviluppo dell'applicazione
 - Più la soluzione presentata è “fantasiosa” e “gradevole agli occhi” più è probabile che il mancato rispetto dei vincoli di progetto possa essere perdonato.
 - Sono state accennate diverse soluzioni implementative: il voto dello studente sarà in funzione di quanto oltre la soluzione presentata vada oltre quella indicata come “baseline”.
- **Posizionamento corretto dei comandi *gl* all'interno delle callback dell'istanza di *GLSurfaceView.Renderer* e minimizzazione dei cambiamenti di stato.**
- Dimostrare di aver sviluppato l'app in maniera autonoma:
 - Lo studente dovrà dimostrare di aver compreso totalmente il codice che porta all'esame.
 - Lo studente dovrà dichiarare eventuali copia-e-incolla e spiegare come sono stati adattati al suo codice.
- Correttezza funzionale e corretta gestione degli errori:
 - L'app tende a crashare senza dare informazioni utili?

Cosa fare e cosa non fare?

- **Si può** utilizzare gli esempi di codice dei Renderer messi a disposizione nel git del corso. Anche del boilerplate code (e.g. ShaderCompiler, PlyParser) e gli assets/risorse.
- **Si può** replicare fedelmente la soluzione di esempio presentata. O si può fare qualcosa di propria inventiva stando attenti ai vincoli imposti.
- **Si può** presentare l'app in un dispositivo reale o virtuale. E' indifferente.
- **NON si può** utilizzare librerie di terze parti (fuori dall'ambito Google Android Dev. Kit) senza prima averlo concordato con il docente.
- **NON si può** utilizzare dei game engine o qualsiasi altra astrazione di livello più alto che i bindings di OpenGL ES.
- **NON si può** utilizzare funzioni *gl* o GLSL built-in keywords deprecated (GL ES version < 2.00) e/o rimosse sia lato host, che lato device. Quando si è in dubbio, controllare qui: <http://web.eecs.umich.edu/~sugih/courses/eecs487/common/notes/APITables.xml> e https://www.slideshare.net/Khronos_Group/opengl-es-32-reference-guide