

Efficient Voxelization

Using Projected Optimal Scanline

François Palma
Nicolas Marguerit
Alexandre Réaubourg

January 11, 2022

Sommaire

1. État de l'art
2. Etude de l'article
3. Prototypage
4. Résultats & Benchmark

État de l'art

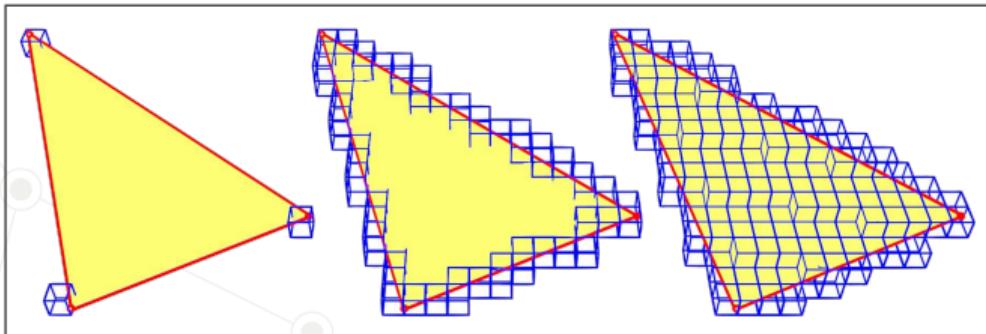
Voxelization

Exemple d'utilisation:

- ▶ Modélisation
- ▶ Simulation physique
 - ▶ De nouvelles simulations rendues possibles
- ▶ Eclairage volumétrique

Etude de l'article

Spécificité de l'algorithme



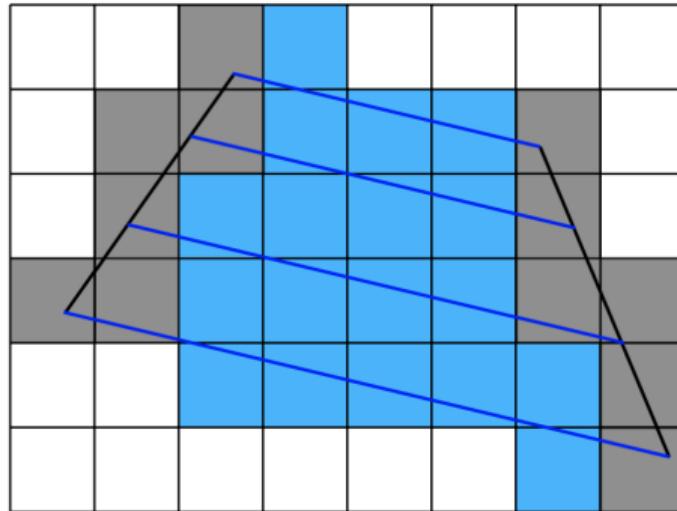
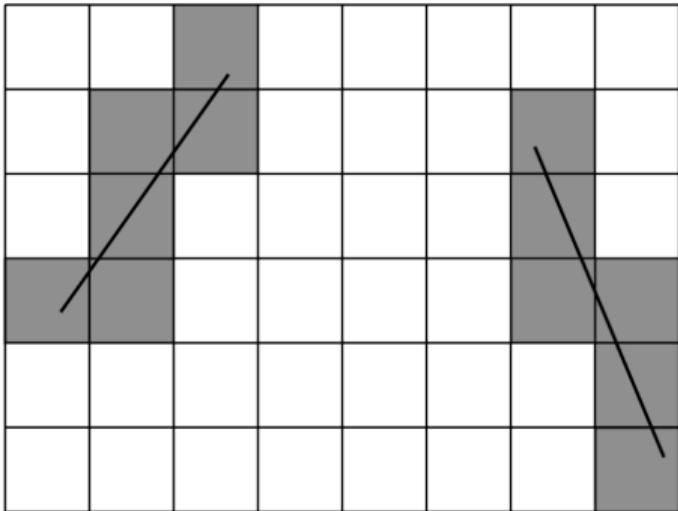
Etapes de la voxelization

Algorithm 1 Triangle Voxelization

```
1: function VOXELIZETRIANGLE( $v_0, v_1, v_2, n$ )
2:    $\{P_0, P_1, P_2\} \leftarrow GETVOXEL(v_0, v_1, v_2)$ 
3:    $i \leftarrow DOMINANTAXISINDEX(P_0, P_1, P_2)$ 
4:   SORTONAXIS( $P_0, P_1, P_2, i$ )
5:   MARKLINEILV( $P_0, P_1, Q_0$ )
6:   MARKLINEILV( $P_1, P_2, Q_1$ )
7:   MARKLINEILV( $P_0, P_2, Q_2$ )
8:    $Q_1 \leftarrow Q_0 \cup Q_1$ 
9:   FILLINTERIOR( $Q_1, Q_2, P_0, P_2, i$ )
10:
11: function MARKLINEILV( $P_0, P_1, Q$ )
12:    $\Delta P[0] \leftarrow SIGN(P_1[0] - P_0[0])$ 
13:    $\Delta P[1] \leftarrow SIGN(P_1[1] - P_0[1])$ 
14:    $\Delta P[2] \leftarrow SIGN(P_1[2] - P_0[2])$ 
15:    $L[0] \leftarrow M[0] \leftarrow |P_1[1] - P_0[1]| |P_1[2] - P_0[2]|$ 
16:    $L[1] \leftarrow M[1] \leftarrow |P_1[0] - P_0[0]| |P_1[2] - P_0[2]|$ 
17:    $L[2] \leftarrow M[2] \leftarrow |P_1[0] - P_0[0]| |P_1[1] - P_0[1]|$ 
18:    $P_{current} \leftarrow P_0$ 
19:   while  $P_{current} \neq P_1$  do
20:      $(L_{min}, L_{index}) \leftarrow MIN(L[0], L[1], L[2])$ 
21:      $P_{current}[L_{index}] \leftarrow P_{current}[L_{index}] + \Delta P[L_{index}]$ 
22:      $L \leftarrow L - L_{min}$ 
23:      $L[L_{index}] \leftarrow 2M[L_{index}]$ 
24:     MARKVOXEL( $P_{current}$ )
25:      $Q.PUSHBACK(P_{current})$ 
26:
27: function FILLINTERIOR( $Q_1, Q_2, P_0, P_2, axis$ )
28:   for  $i = 0$  to  $P_2[axis] - P_0[axis]$  do
29:     slice  $\leftarrow P_0[axis] + i + 1/2$ 
30:      $Q_{1,sub} \leftarrow GETSUBSEQUENCE(Q_1, slice)$ 
31:      $Q_{2,sub} \leftarrow GETSUBSEQUENCE(Q_2, slice)$ 
32:     while  $Q_{1,sub} \neq \emptyset$  OR  $Q_{2,sub} \neq \emptyset$  do
33:        $P_{start} \leftarrow GETNEXTINSlice(Q_{1,sub})$ 
34:        $P_{stop} \leftarrow GETNEXTINSlice(Q_{2,sub})$ 
35:       MARKLINEILV( $P_{start}, P_{stop}$ )
```

Etude de l'article

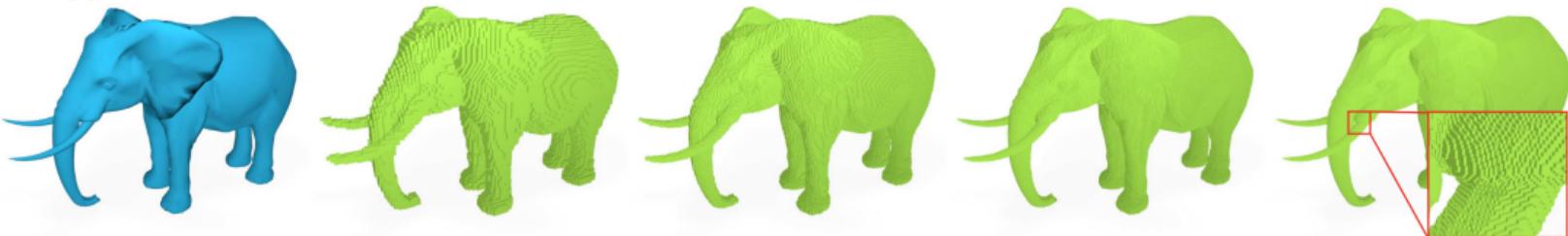
3D voxelization ? scanlines ?



Etude de l'article

Points clés

- ▶ Optimisation en temps
- ▶ Parfaitement compatible avec le multithreading
- ▶ Plus on a de triangles plus c'est relativement rapide
- ▶ la discréétisation par des entiers réduit les trous dans la couverture



Prototypage

Notre plan initial

- ▶ Voxelization d'un triangle
- ▶ Utiliser OpenGL
- ▶ Prototypage 2D

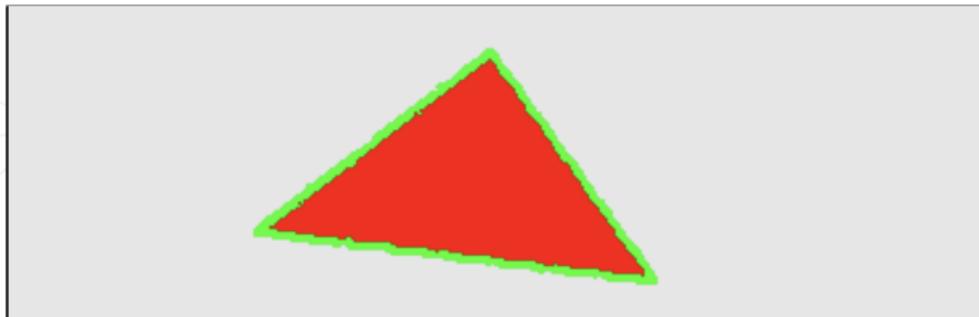


Nous pensions que tout était simple et clair

Prototypage

Premiers jours

- ▶ Reprise en main d'OpenGL
- ▶ Très satisfait du rendu du triangle à l'écran
- ▶ Voxelization des côtés se fait rapidement
- ▶ Parties qui ne sont pas claires dans l'algo

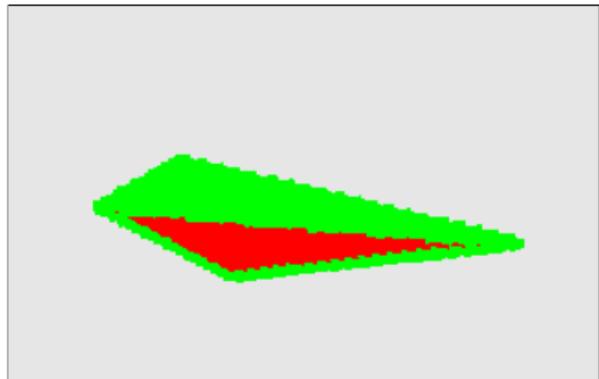


Voxelization des côtés

Prototypage

Difficultés rencontrées

- ▶ Débogage du problème de la pyramide
- ▶ Essayer la méthode naïve et se rendre compte qu'elle marche mieux
- ▶ Réaliser que l'algo est incomplet



Prototypage

Éclaircissement des zones d'ombres sur l'article

- ▶ Relecture minutieuse de l'article pour trouver où se trouve le problème
- ▶ Manque de détail dans les cas 2D
- ▶ Choix d'utilisation de Bresenham dans les cas 2D

Résultats & Benchmark

Quelques chiffres en comparaison

	Velocity	Angle	Vertical force
	U [m/s]	α [°]	F_z [N]
2D simulation	9	2	9.23
3D simulation	10.0	3	15.039
Experiment A	11.31	2.5	13.2
Experiment B	11.26	2.7	12.6
Experiment C	11.33	2.47	13.6

Résultats & Benchmark

Démo



Résultats & Benchmark

Perspectives d'évolution

- ▶ Utiliser CUDA
- ▶ Changer de langage
- ▶ Passer sur du multithread

Résultats & Benchmark

Conclusion

Implémentation de l'algo de voxelization de l'article en OpenGL réussi mais avec notre interprétation pour le problème des cas 2D