Aplicações para geração de vértices em GPU

Gustavo Bastos Nunes

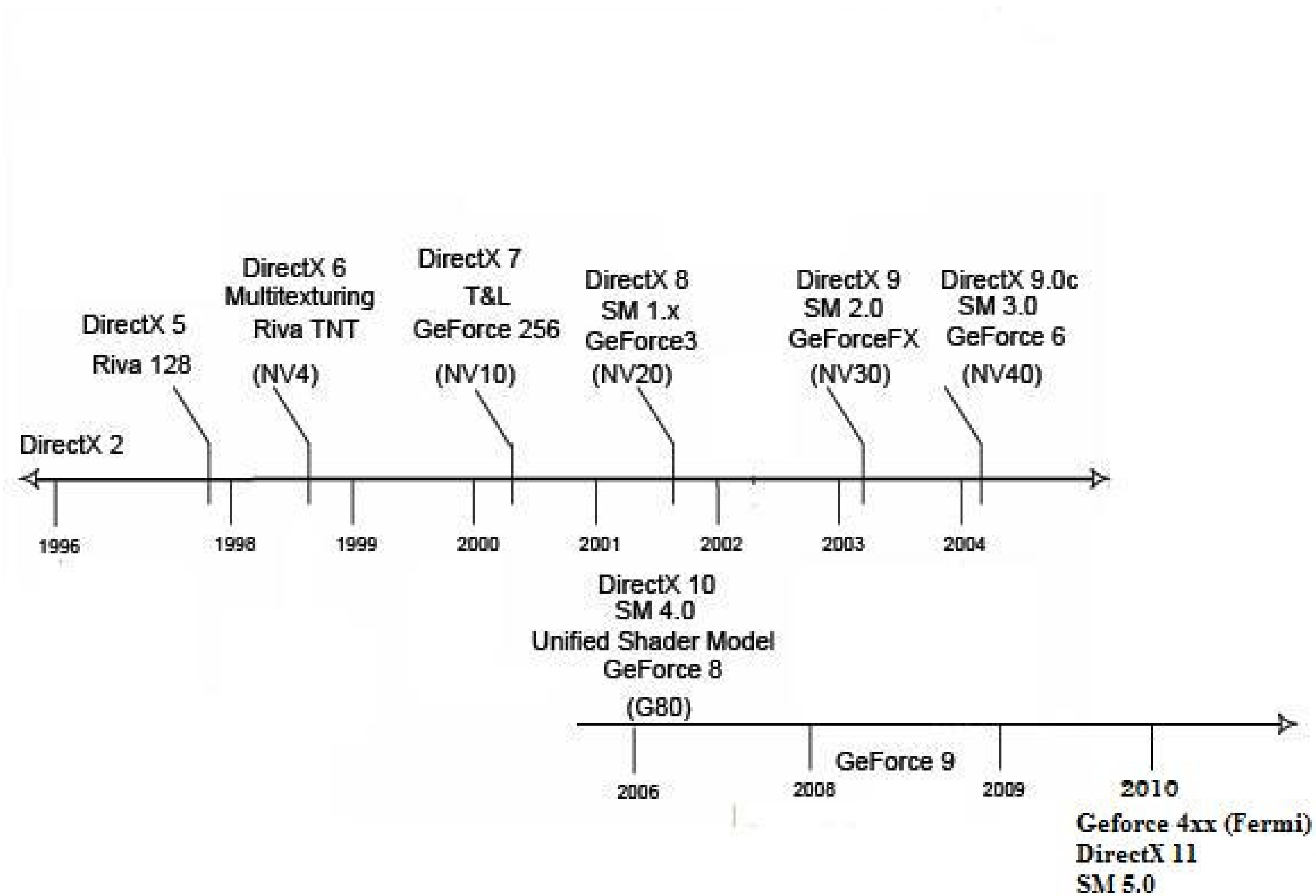
Orientador: Alberto Barbosa Raposo

Co-orientador: Bruno Feijó

Conteúdo

- Introdução – História do pipeline
- Motivação
- Performance do Tessellator
- PN-Triangles vs Phong Tessellation
- Renderização de Tubos 3D
- Renderização de Terrenos em GPU
- Bibliografia

História do pipeline



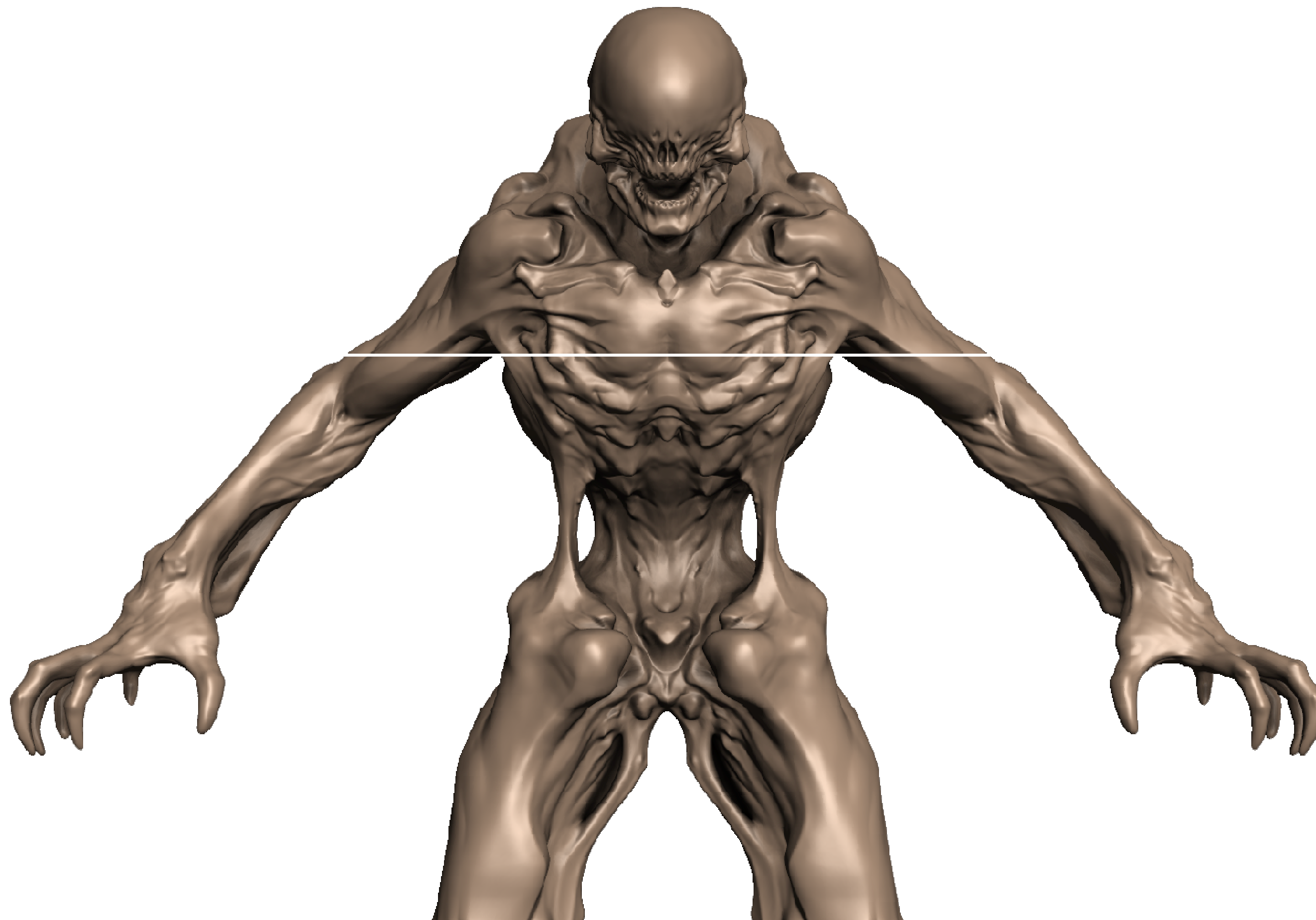
Motivação

- modelos com alto nível de detalhe
- redução da necessidade de largura de banda
- animação mais eficiente
  - só precisa animar a malha grosseira
  - subdivide após a animação
- view-dependent LOD contínuo
- Possibilidade de novos algoritmos

Motivação

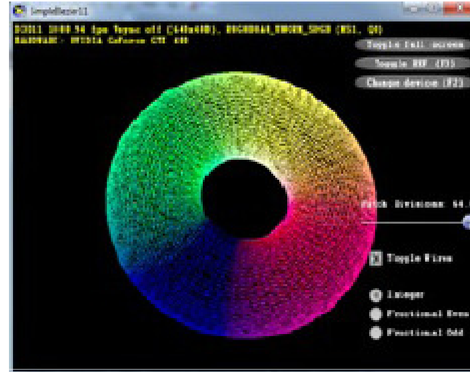
Silhueta facetada no jogo MAFIA II

Motivação



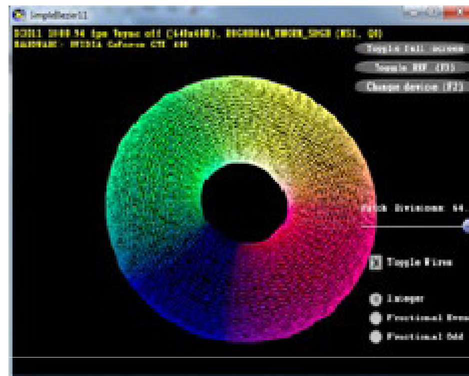
Silhuetas bem delineadas com o Tessellator

Performance do Tessellator



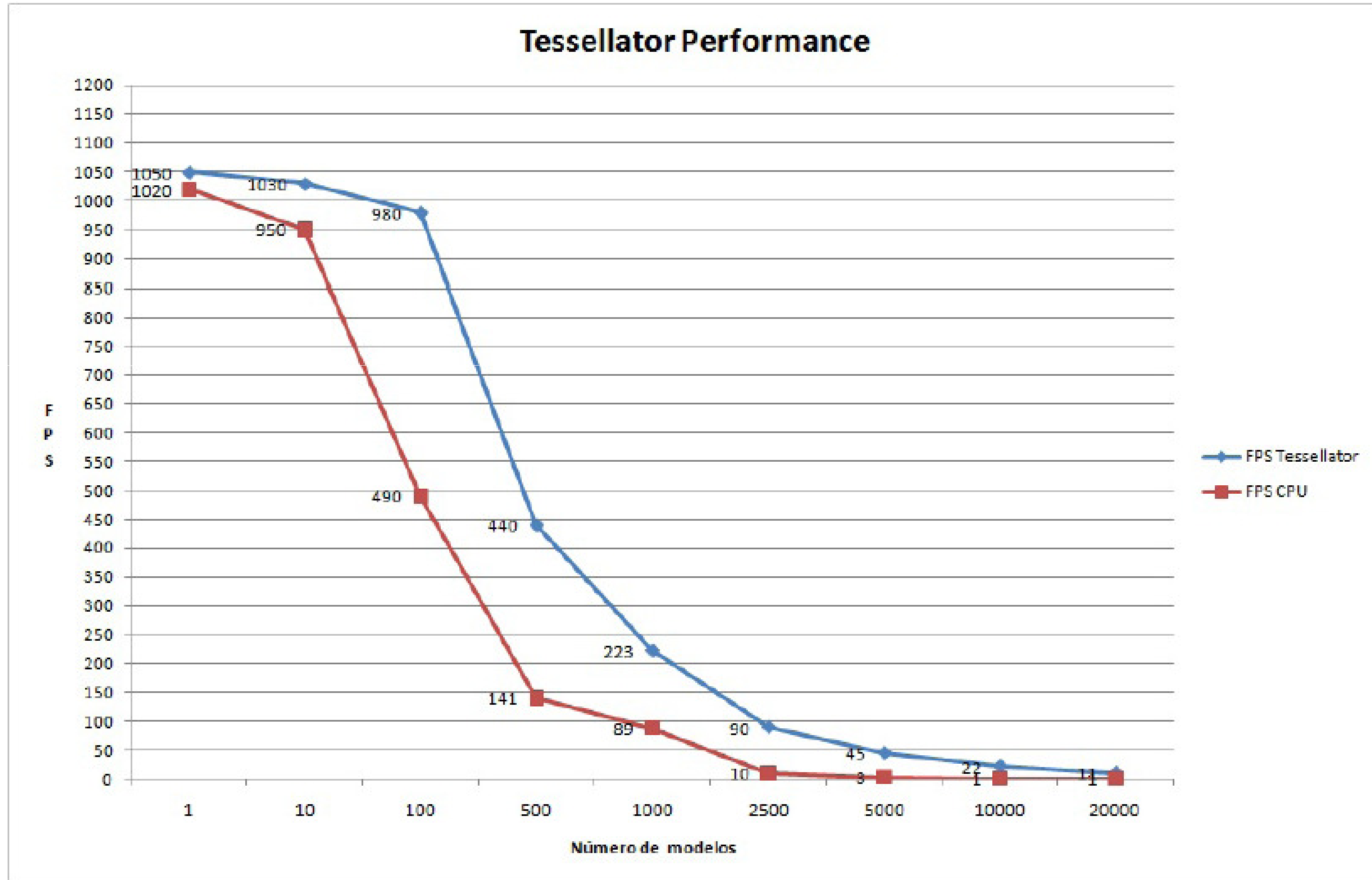
Quantidade de Torus	Triângulos(em milhões)	FPS Tessellator	FPS CPU
20000	163.84	11	1
10000	81.92	22	1
5000	40.96	45	3
2500	20.48	90	10
1000	8.192	223	89
500	4.096	440	141
100	0.8192	980	490
10	0.08192	1030	950
1	0.008192	1050	1020

Performance do Tessellator

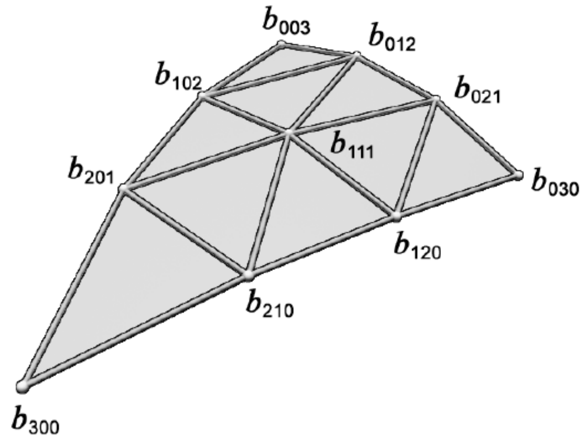


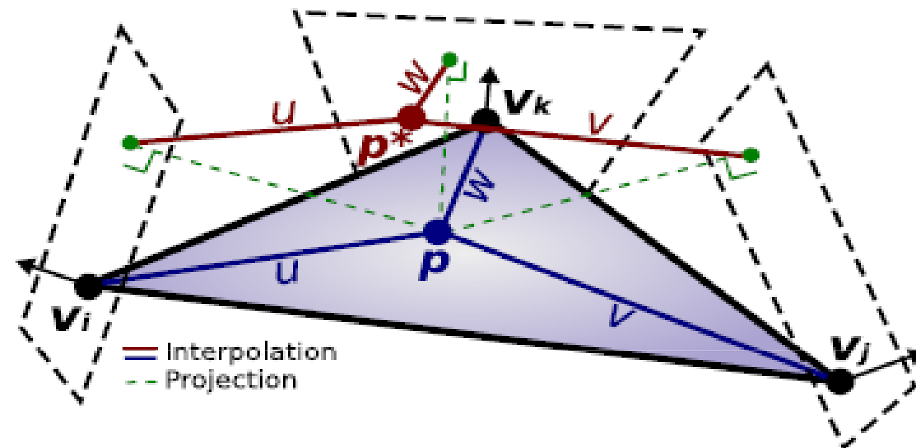
CPU Memory	GPU Memory
5898.24	0.24
2949.12	0.12
1479.56	0.06
737.28	0.03
294.91	0.012
147.45	0.006
29.49	0.0012
2.94	0.00012
0.29	0.000012

Performance do Tessellator



PN-Triangles vs Phong Tessellation

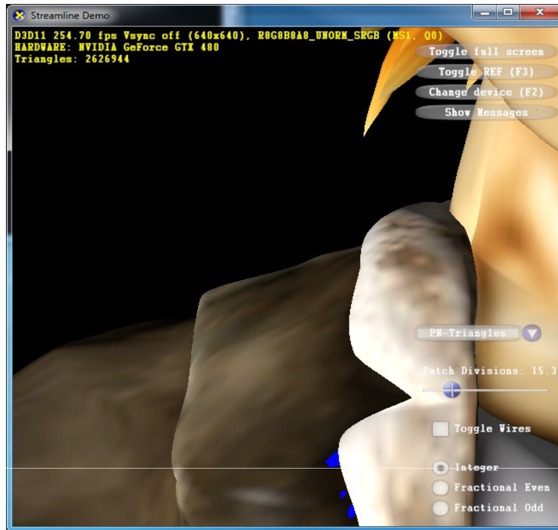


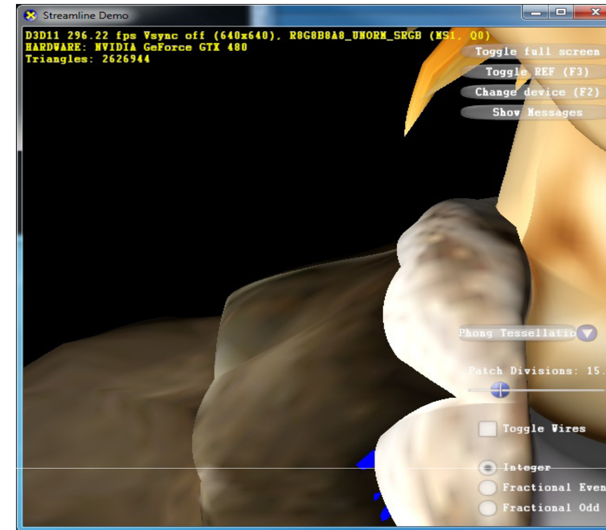


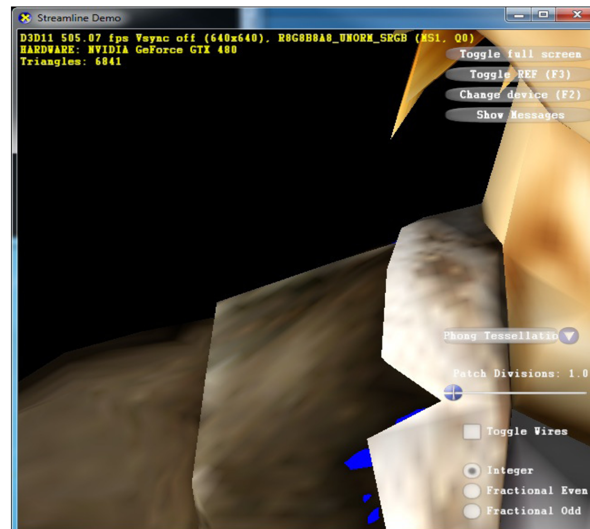
$b : \Re^2 \rightarrow \Re^3$, para $w = 1 - u - v,\ u, v, w \geq 0$

$$\begin{aligned} b(u,v) &= \sum_{i+j+k=3} b_{ijk} \frac{3!}{i!j!k!} u^i v^j w^k, \\ &= b_{300}w^3 + b_{030}u^3 + b_{003}v^3 \\ &+ b_{210}3w^2u + b_{120}3wu^2 + b_{201}3w^2v \\ &+ b_{021}3u^2v + b_{102}3wv^2 + b_{012}3uv^2 \\ &+ b_{111}6wuv. \end{aligned}$$

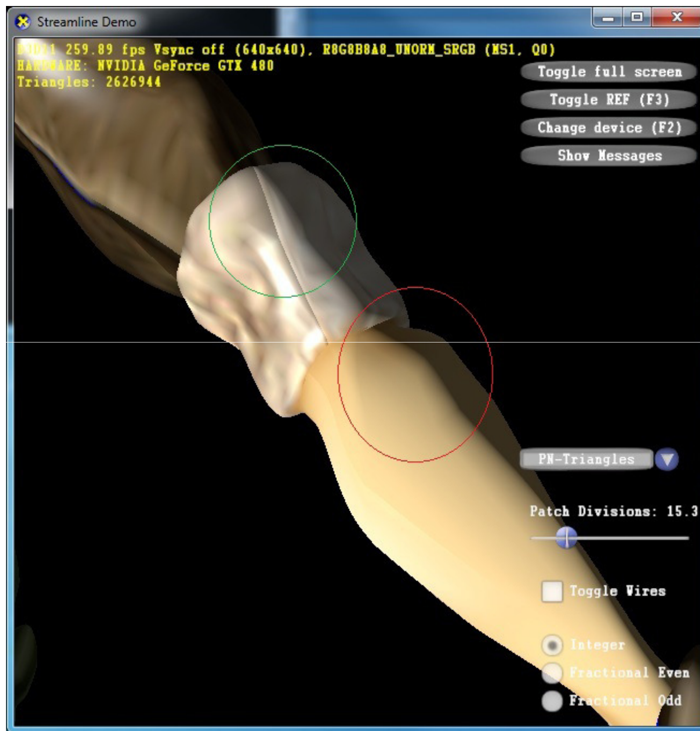
PN-Triangles vs Phong Tessellation



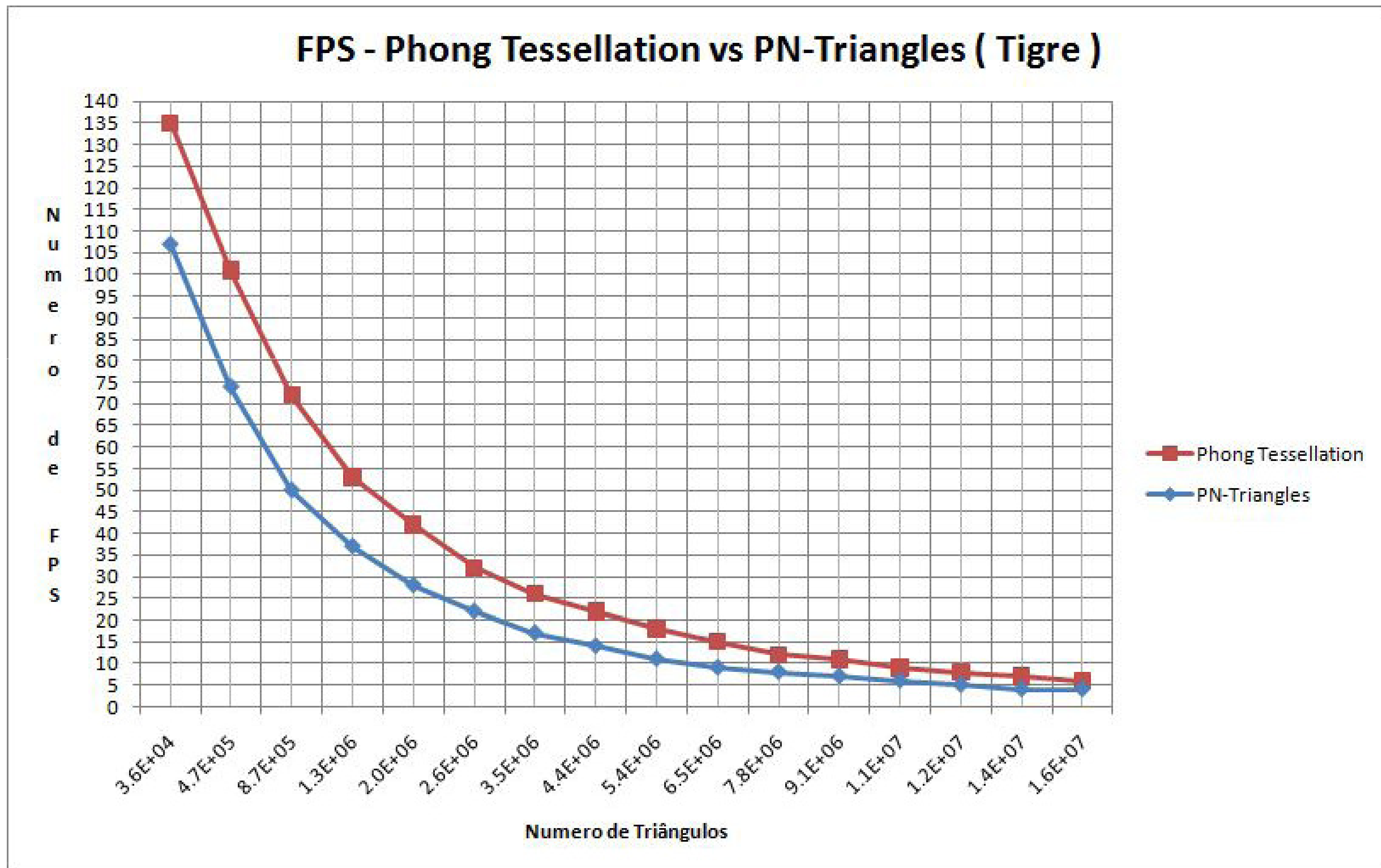




PN-Triangles vs Phong Tessellation



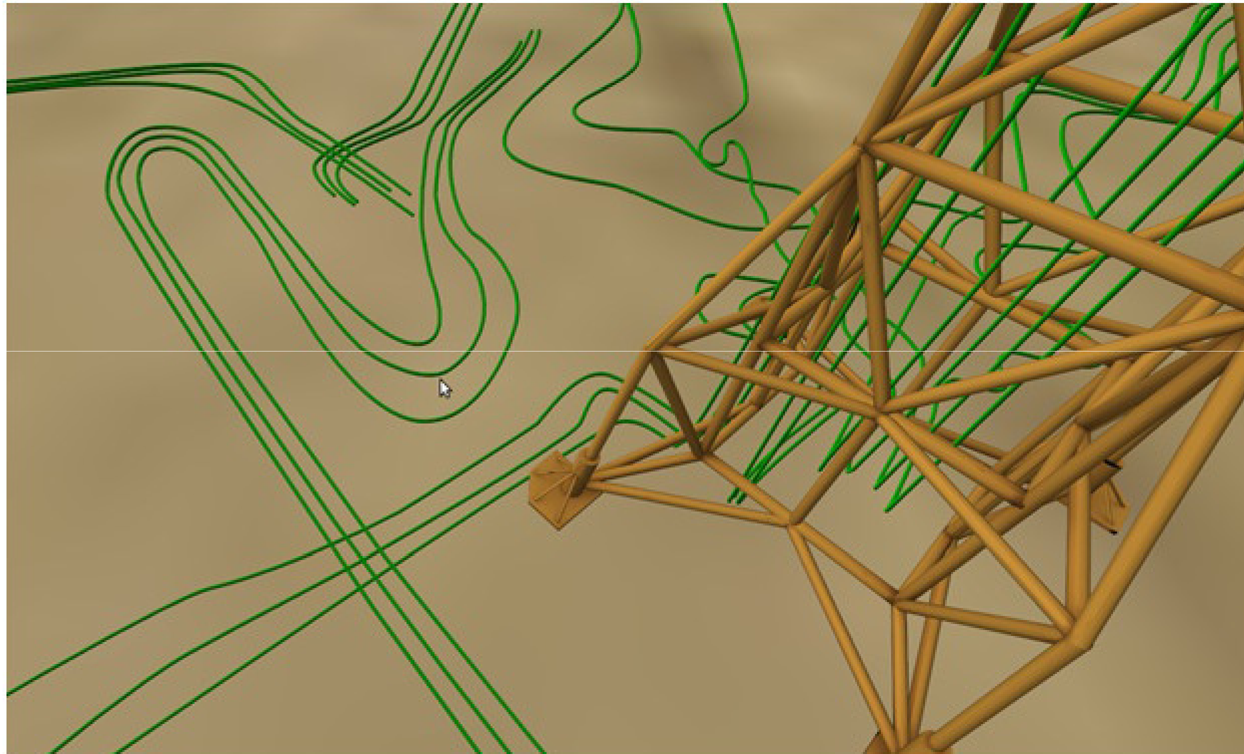
PN-Triangles vs Phong Tessellation



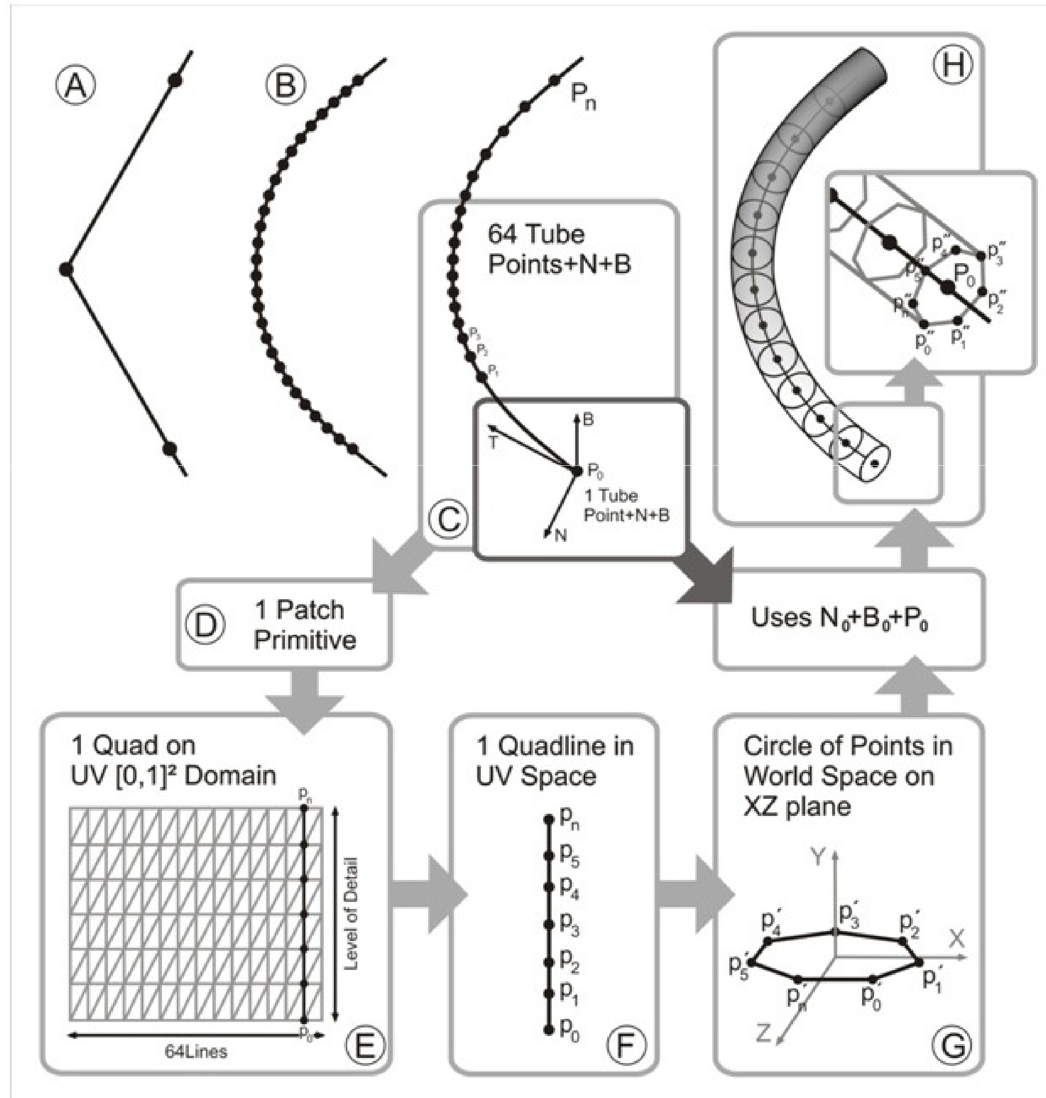
PN-Triangles vs Phong Tessellation

Tigre (Número de Triângulos em milhões)	Ganho (em %)	Ganho (em FPS)
0.036	26.17%	28
0.47	36.49%	27
0.87	44.00%	22
1.30	43.24%	16
2.00	50.00%	14
2.60	45.45%	10
3.50	52.94%	9
4.40	57.14%	8
5.40	63.64%	7
6.50	66.67%	6
7.80	50.00%	4
9.1	57.14%	4
11.0	50.00%	3
12.0	60.00%	3
14.0	75.00%	3
16.0	50.00%	2

Renderização de Tubos 3D

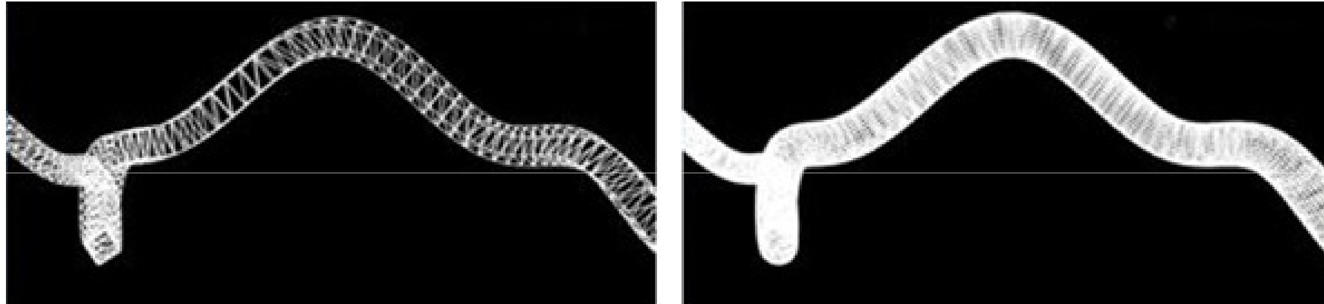


Renderização de Tubos 3D



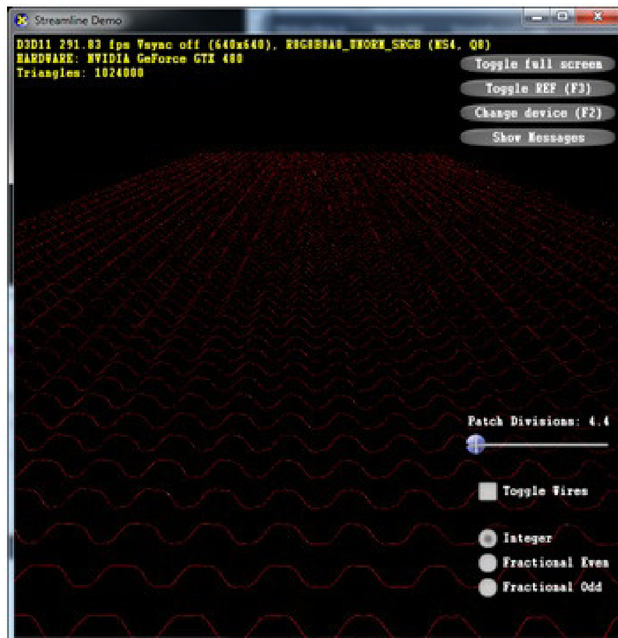
Renderização de Tubos 3D

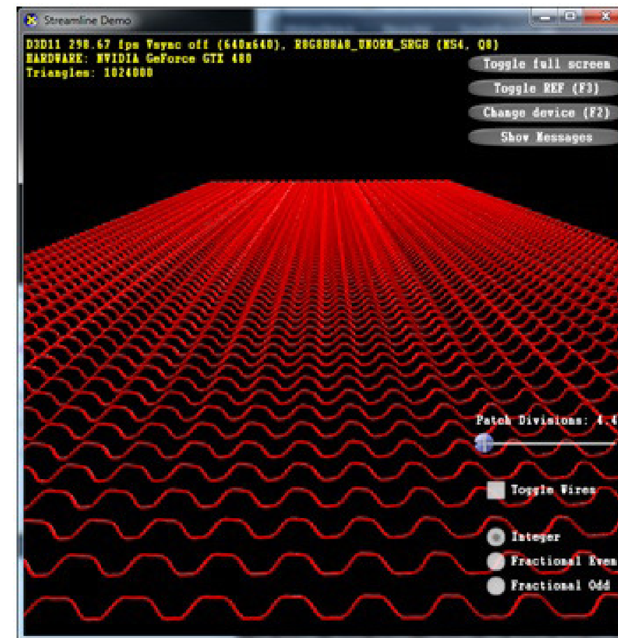
- Possibilidade de LOD contínuo



Renderização de Tubos 3D – Melhora do aliasing

- Raio pode ser setado dinamicamente por patch no Hull Shader
- Em nossa solução setamos o raio para ocupar dois pixels em espaço de mundo.





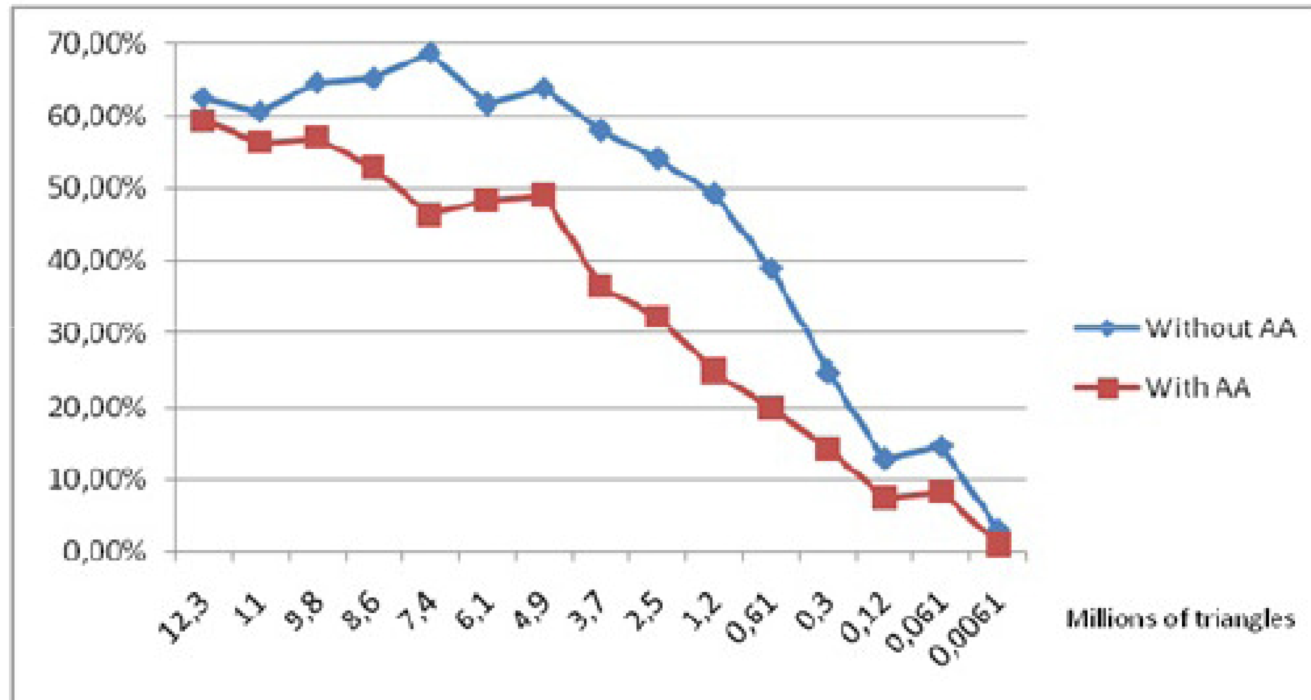
Renderização de Tubos 3D

Milhões de Triângulos	FPS - técnica CPU	FPS - Tessellator com AA	FPS - Tessellator sem AA
184,3	0	10	11
28,6	1	59	60
12,3	64	102	104
11	71	111	114
9,8	79	124	130
8,6	89	136	147
7,4	99	145	167
6,1	120	178	194
4,9	141	210	231
3,7	181	247	286
2,5	242	320	373
1,2	335	418	500
0,61	518	621	720
0,3	730	835	911
0,12	930	999	1050
0,061	970	1050	1112
0,0061	1100	1111	1135

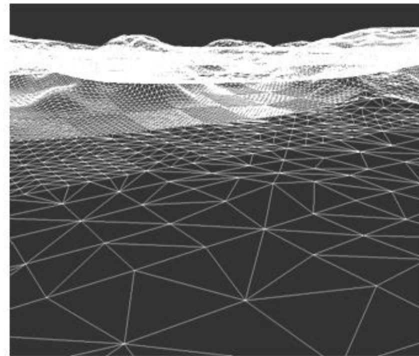
Renderização de Tubos 3D

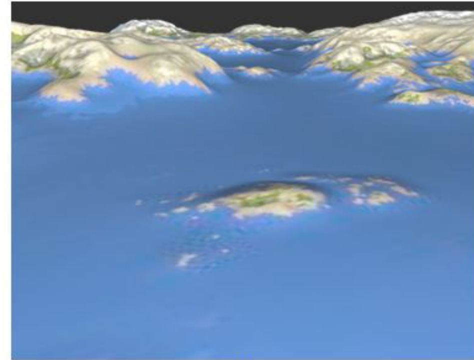
Memória técnica CPU (em MB)	Nossa técnica(em MB)
13270	184,32
2059	28,8
886	12,16
792	10,88
706	9,6
619	8,32
533	7,68
439	6,4
353	5,12
266	3,84
180	2,56
86	1,28
44	0,64
22	0,32
9	0,16
4	0,08
0	0,00

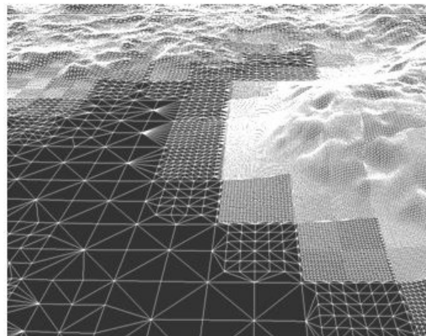
Renderização de Tubos 3D



Renderização de Terrenos em GPU


(a)

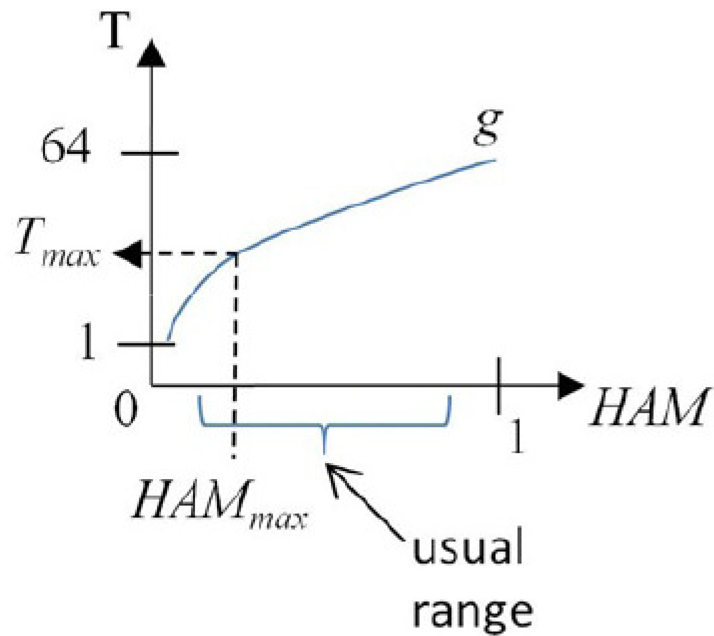

(b)

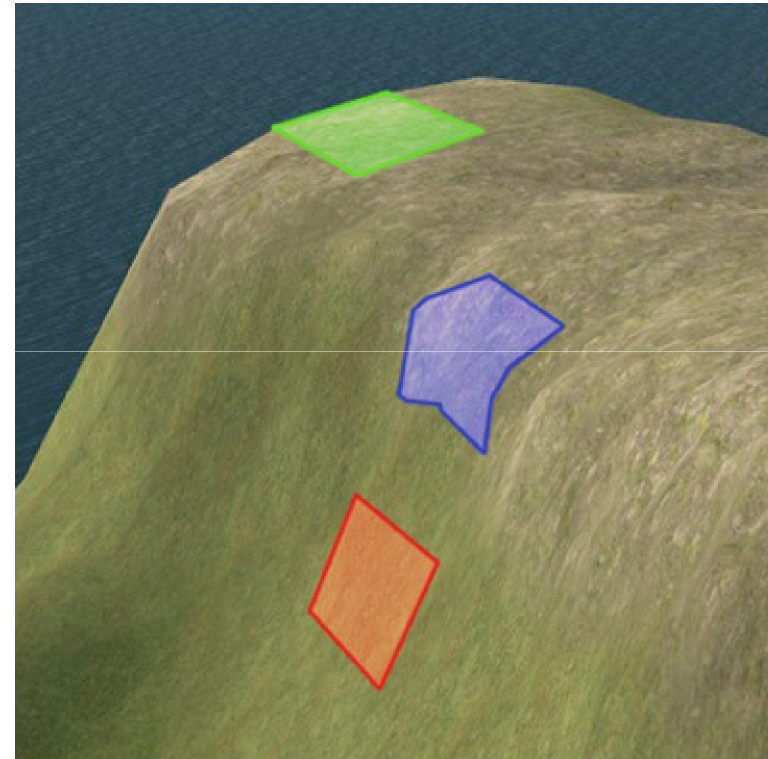

(c)

(d)

- Adicionado Frustum Culling em GPU em relação ao artigo original

Renderização de Terrenos em GPU – Tecelagem máxima(HAM)

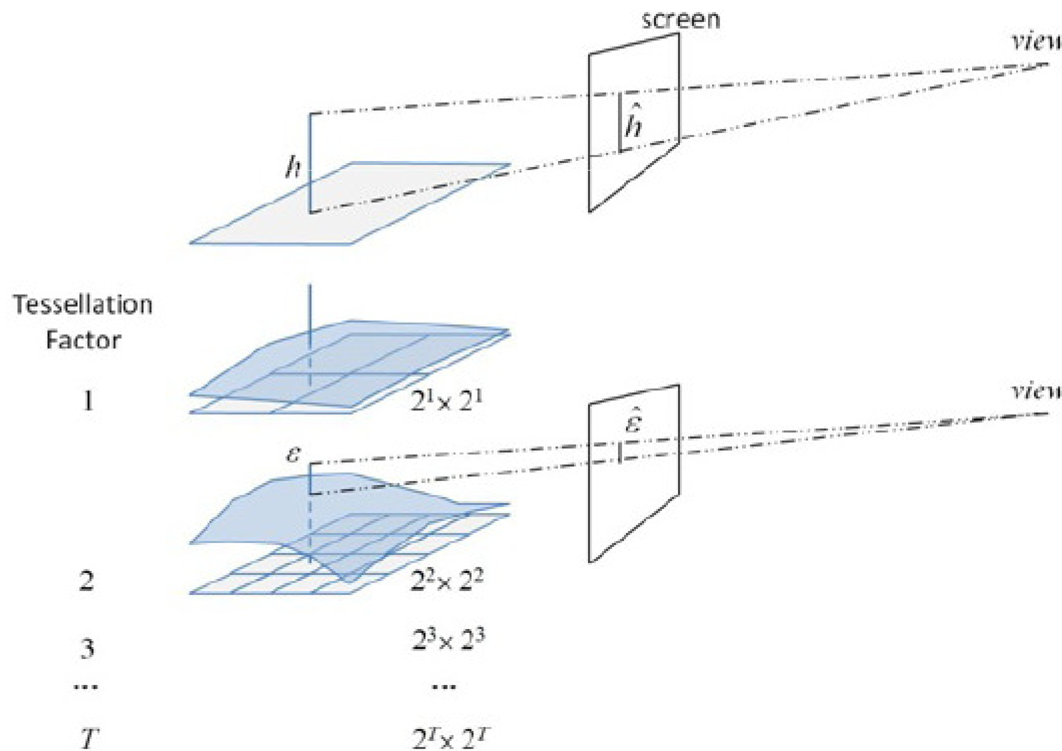




$$T_{max} = g(HAM_{max})$$

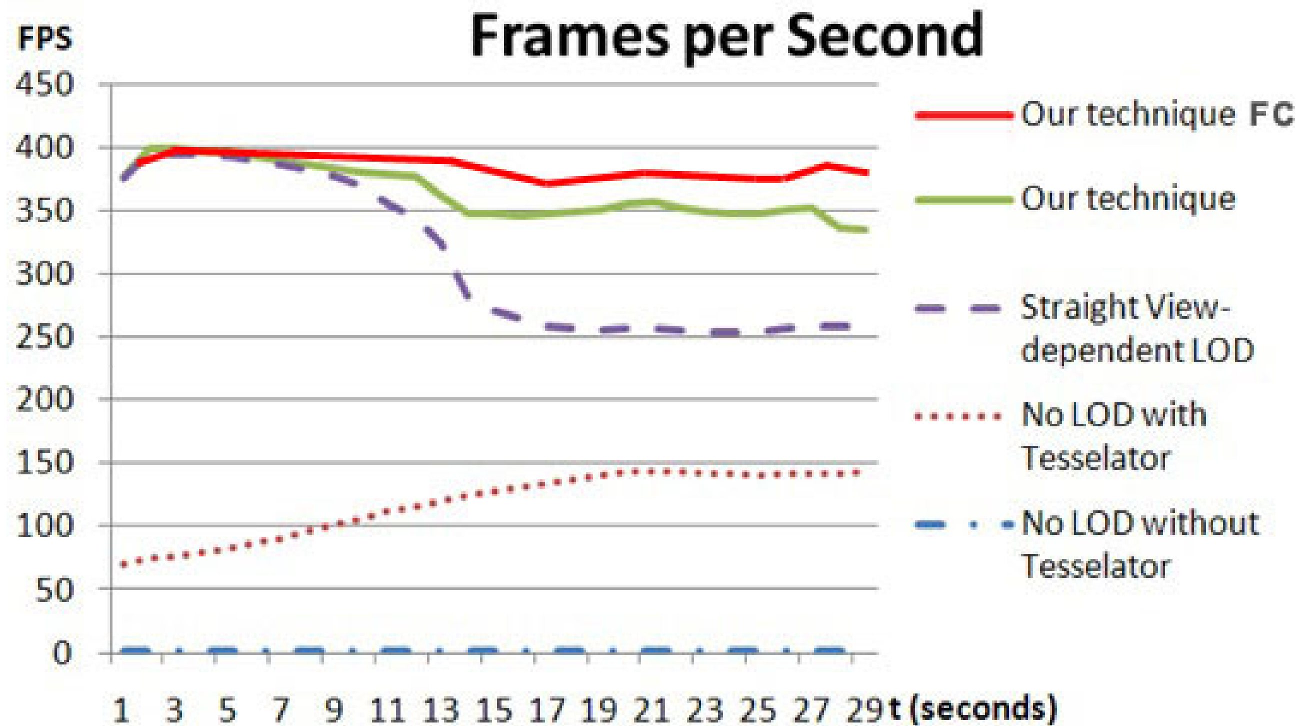
Renderização de Terrenos em GPU – Tecelagem mínima

- Razão entre a tolerância e a altura em espaço de tela é proporcional a 2^T

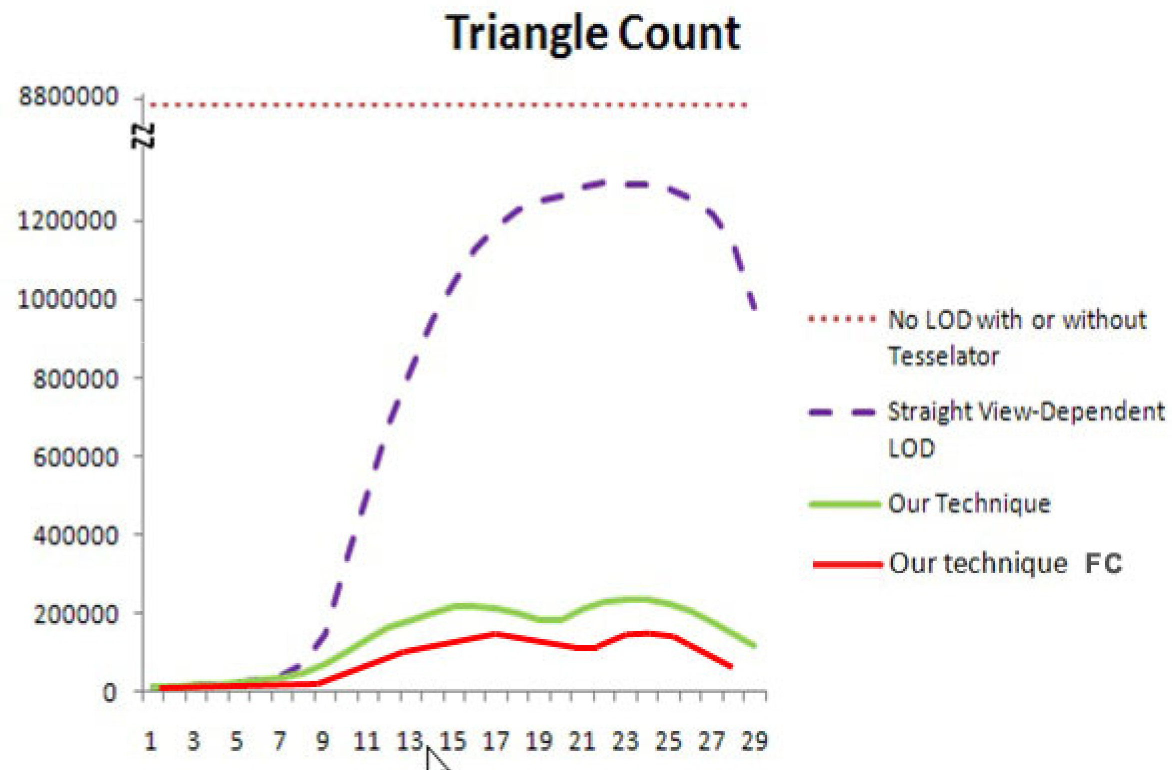


$$T_{min} = \log_2 \epsilon / \hat{h}$$

Renderização de Terrenos em GPU

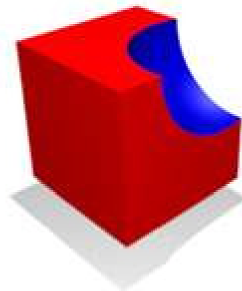


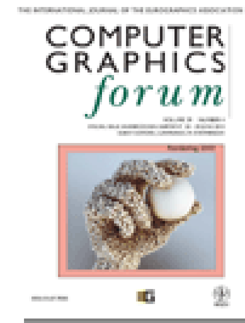
Renderização de Terrenos em GPU



Outras possibilidades

- J.C. Yang, J. Hensley, H. Grün, and N. Thibieroz, "Real-Time Concurrent Linked List Construction on the GPU", presented at Comput. Graph. Forum, 2010, pp.1297-1304.
- Order Independent Transparency





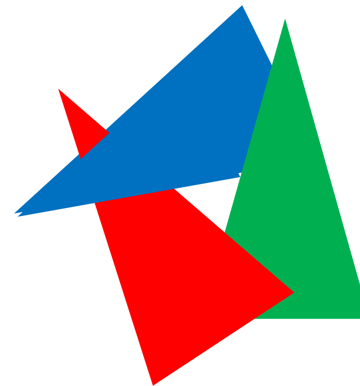
Computer Graphics Forum

Volume 29, Issue 4, pages 1297–1304, June 2010

Order Independent Transparency

Construction by Example

- Classical problem in computer graphics
- Correct rendering of semi-transparent geometry requires sorting – blending is an order dependent operation
- Sometimes sorting triangles is enough but not always
  - Difficult to sort: Multiple meshes interacting (many draw calls)
  - Impossible to sort: Intersecting triangles (must sort fragments)



Try doing this in PowerPoint!

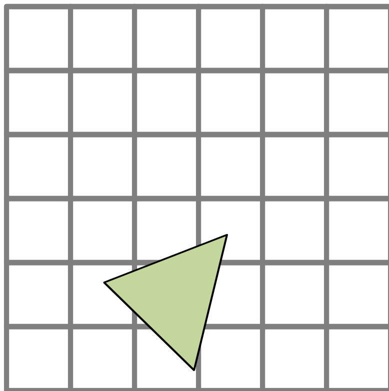
Algorithm Overview

0. Render opaque scene objects
1. Render transparent scene objects
2. Screen quad resolves and composites fragment lists

Step 0 – Render Opaque

- Render all opaque geometry normally

Render Target



Algorithm Overview

0. Render opaque scene objects
1. Render transparent scene objects
   - All fragments are stored using per-pixel linked lists
   - Store fragment's: color, alpha, & depth
2. Screen quad resolves and composites fragment lists

Setup

- Two buffers
  - Screen sized head pointer buffer
  - Node buffer – large enough to handle all fragments

- Render as usual

Step 1 – Create Linked List

Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

Render Target

Counter = 0

Node Buffer

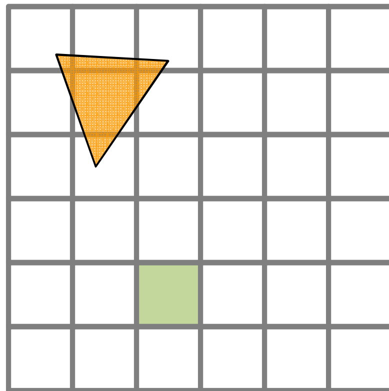
0 1 2 3 4 5 6 …

Step 1 – Create Linked List

Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

Render Target



Counter = 0

Node Buffer

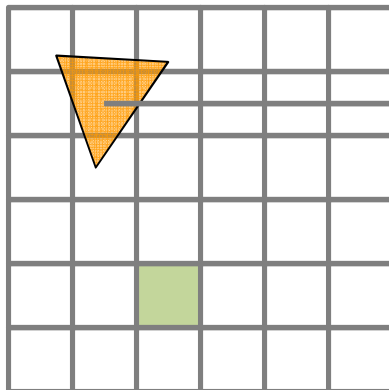
0 1 2 3 4 5 6 …

Step 1 – Create Linked List

Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

Render Target



Counter = 1

Node Buffer

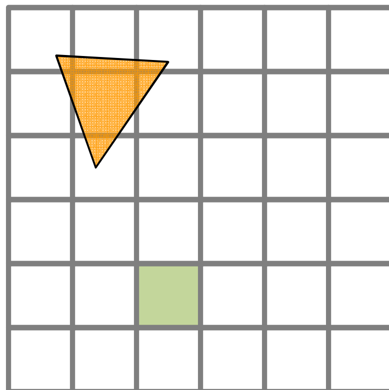
0 1 2 3 4 5 6 …

Step 1 – Create Linked List

Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	0	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1

Render Target



Counter = 1

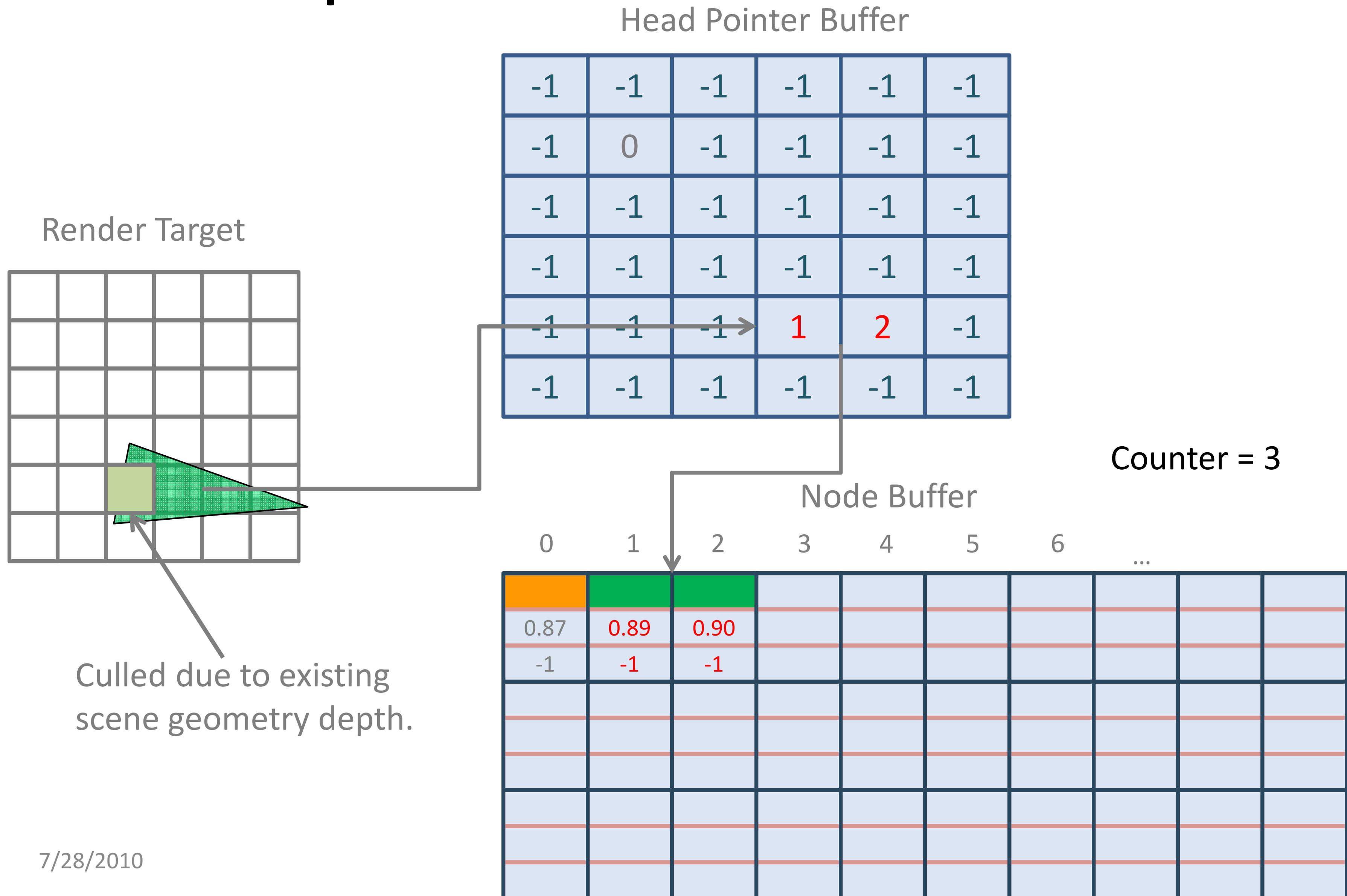
Node Buffer

0 1 2 3 4 5 6 …

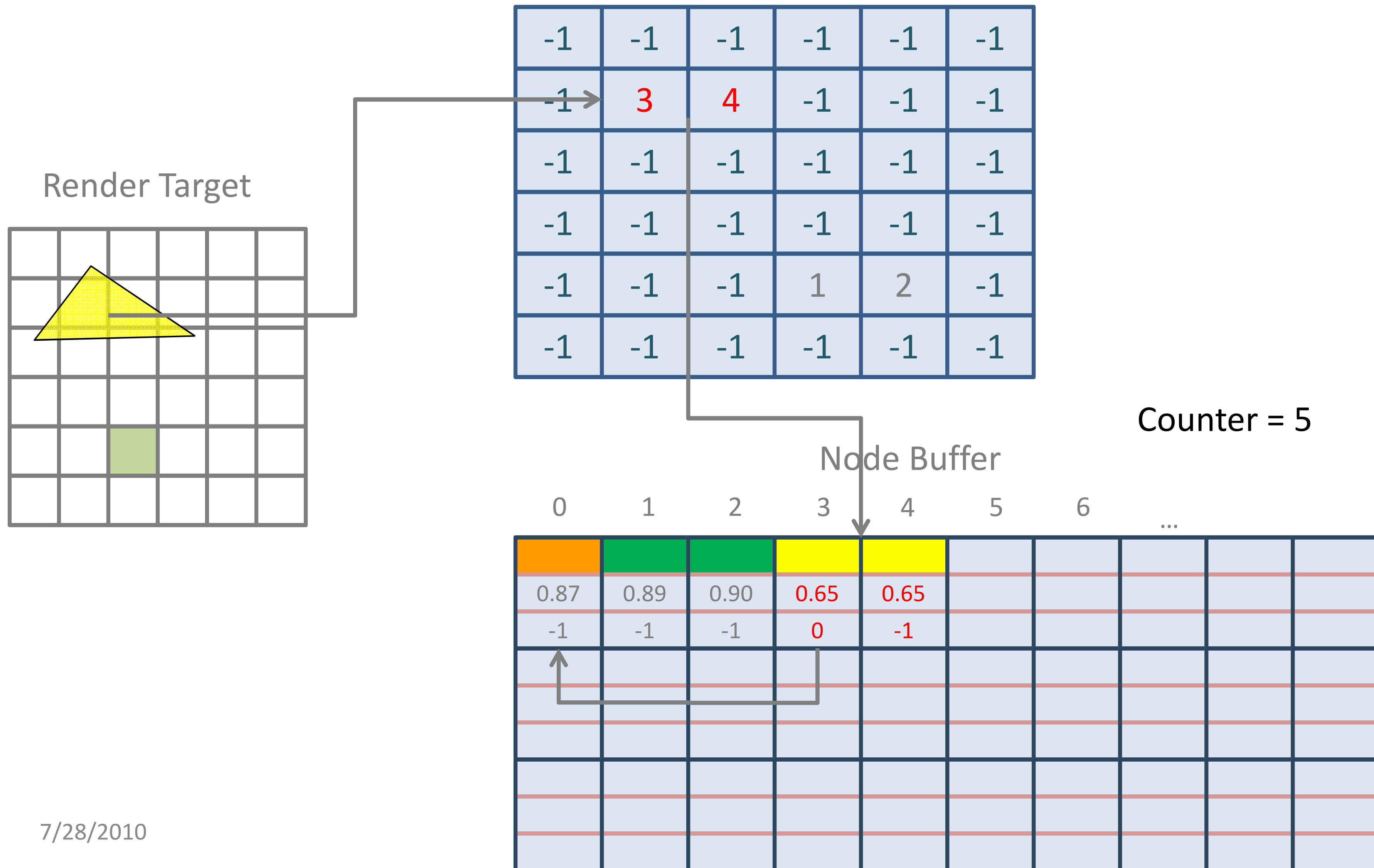
0.87

-1

Step 1 – Create Linked List

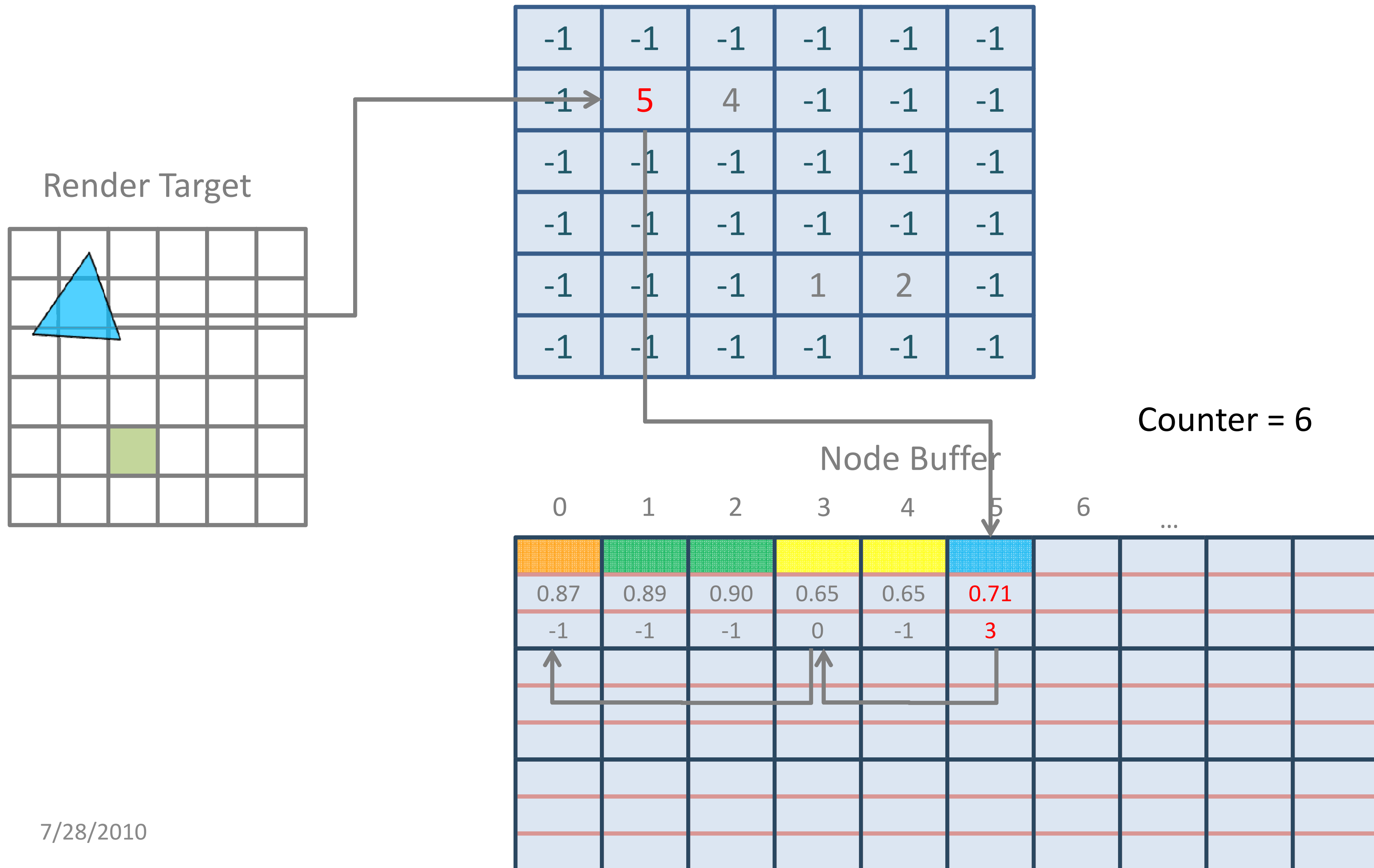


Step 1 – Create Linked List



7/28/2010

Step 1 – Create Linked List



Node Buffer Counter

- Counter allocated in GPU memory (i.e. a buffer)
  - Atomic updates
  - Contention issues
- DX11 Append feature
  - Linear writes to a buffer
  - Implicit writes
    - Append()
  - Explicit writes
    - IncrementCounter()
    - Standard memory operations
  - Up to 60% faster than memory counters

Algorithm Overview

0. Render opaque scene objects
1. Render transparent scene objects
2. Screen quad resolves and composites fragment lists
   - Single pass
   - Pixel shader sorts associated linked list (e.g., insertion sort)
   - Composite fragments in sorted order with background
   - Output final fragment

Step 2 – Render Fragments

Render Target

(0,0)->(1,1):
Fetch Head Pointer: -1
-1 indicates no fragment to render

Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Node Buffer

0	1	2	3	4	5	6	...		
0.87	0.89	0.90	0.65	0.65	0.71				
-1	-1	-1	0	-1	3				

Step 2 – Render Fragments

Head Pointer Buffer

-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Render Target

Node Buffer

0	1	2	3	4	5	6	…
0.87	0.89	0.90	0.65	0.65	0.71		
-1	-1	-1	0	-1	3		

(1,1):
Fetch Head Pointer: 5
Fetch Node Data (5)
Walk the list and store in temp array

0.71	0.65	0.87

Step 2 – Render Fragments

-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Render Target

(1,1):
Sort temp array
Blend colors and write out

0.65	0.71	0.87

Node Buffer

0	1	2	3	4	5	6	...
0.87	0.89	0.90	0.65	0.65	0.71		
-1	-1	-1	0	-1	3		

Step 2 – Render Fragments

Head Pointer Buffer

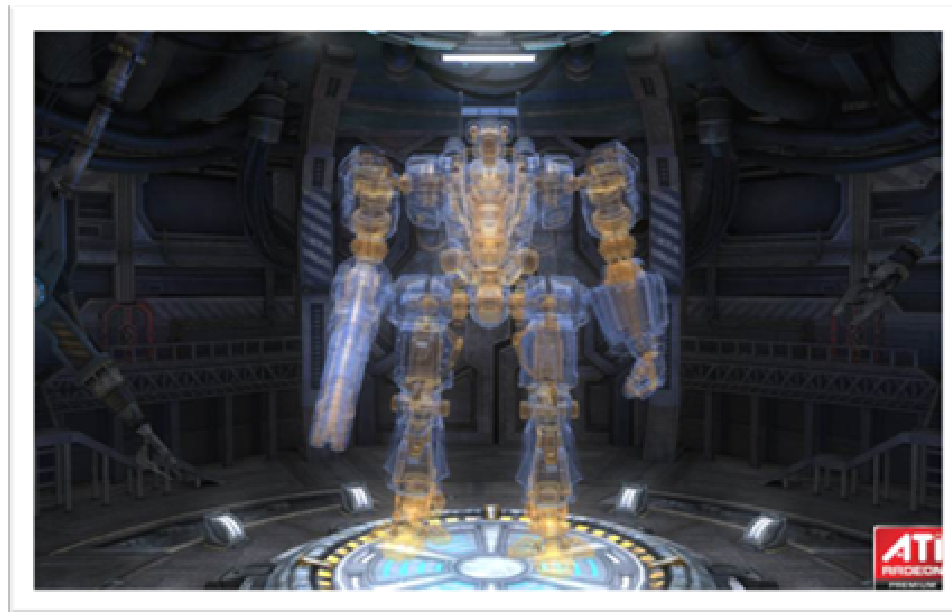
-1	-1	-1	-1	-1	-1
-1	5	4	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	-1	-1	-1
-1	-1	-1	1	2	-1
-1	-1	-1	-1	-1	-1

Render Target

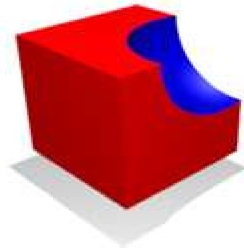
Node Buffer

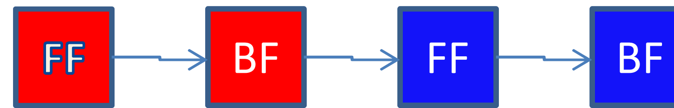
0	1	2	3	4	5	6	…
0.87	0.89	0.90	0.65	0.65	0.71		
-1	-1	-1	0	-1	3		

Demo AMD

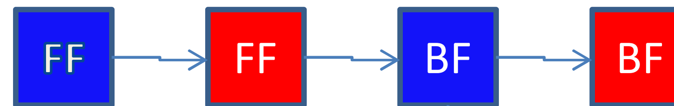


Algoritmo CSG (Subtração)





SORT

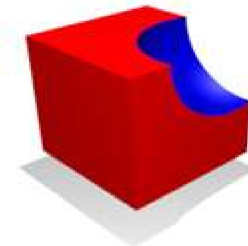


Fragmento a ser renderizado!!
Salva a profundidade

Se perder em Z ou não achar duas FF seguidas, salva a profundidade do primeiro fragmento que não seja da superfície subtraída.

Algoritmo CSG (Subtração)

- Renderiza os objetos marcando se os fragmentos são back ou front faced e a qual superfície ele pertence (a superfície que sofre a subtração ou a superfície subtraída)
- Guarda todos os fragmentos em uma lista encadeada
- Renderiza um screen quad e ordena a lista de fragmentos de frente para trás
- Se o primeiro fragmento da lista for da superfície subtraída significa que ela não perde em Z e o fragmento que deve ser renderizado é o backfaced daquela superfície. Para achar este fragmento, itera na lista encadeada quando tiver passado por dois fragmentos front-faced o próximo é o fragmento desejado.
- Salva a profundidade de cada fragmento a ser renderizado em uma textura.
- Renderiza a cena de novo consultando a textura de profundidade, se o fragmento estiver na profundidade desejada, renderiza ele. Else clip(-1).



Bibliografia

- J. D. Owens, M. Houston, D. Luebke, S. Green, J. E. Stone and J. C. Phillips. *GPU Computing.* Proceedings of the IEEE, 96(5), May 2008.
- Boubekeur, T., and Alexa, M. 2008. Phong tessellation. *ACM Trans. Graph. 27*, 5, 1–5.
- Vlachos, A., Peters, J., Boyd, C., and Mitchell, J. 2001. Curved PN triangles. *Symposium on Interactive 3D Graphics*, 159-166.
- L. Piegl and W. Tiller, *The NURBS Book*, 2nd ed, Berlin, Germany: Springer-Verlag, 1996. Monographs in Visual Communication.
- Loop C., Schaefer S., Ni T., Castaño I.: Approximating Subdivision Surfaces with Gregory Patches for Hardware Tessellation. ACM Transactions on Graphics, 28, 5 (2009), 1–9. 2, 9
- C. Loop. Smooth subdivision surfaces based on triangles. Master's thesis, Dept. of Mathematics, University of Utah, August 1987.
- Loop, C., AND Schaefer, S. 2008. Approximating catmull-clark subdivision surfaces with bicubic patches. ACM Transaction on Graphics 27, 1, 1–8.
- Stoll, C., Gumhold, S., Seidel, H.-P. 2005. Visualization with stylized line primitives. In Silva, C. T., Grller, E., Rushmeier, H., editors, *IEEE Visualization 2005 (VIS 2005),* pages 695-702, Minneappolis, USA. IEEE.
- Merhof, D., Sonntag, M., Enders, F., Nimsky, C., Greiner, G., 2006. Hybrid visualization for white matter tracts using triangle strips and point sprites. In *IEEE Transactions on Visualization and Computer Graphics,* 12(5):1181-1188. Member-Peter Hastre-iter.
- Bloomenthal, J. 1990. *Calculation of reference frames along a space curve. Graphics Gems,* 567-571.
- Toledo, R. "Visualisation Interactive de Modeles Complexes avec les Cartes Graphiques Programmables"[PhD thesis]. INRIA Nancy, Université Henri Poincaré, 2007.
- Ebert D., Musgrave F. K., Peachey D., Perlin K., Worley S., Mark B., and Hart J. *Texture & Modeling: A Procedural Approach, 3rd Edition*. Morgan Kaufmann, 2002.
- Perlin, K. An Image Synthesizer. Proceedings of SIGGRAPH 85. In *Computer Graphics* (1985), vol. 19, ACM SIGGRAPH, pp. 287–296.
- Valdetaro, A., Nunes, G., Raposo, A., Feijó, B., de Toledo, R. LOD terrain rendering by local parallel processing on GPU. In *IX Brazilian Symposium on Computer Games and Digital Entertainment, 2010.*
- Hugues Hoppe*. Progressive meshes. In SIGGRAPH '96 Proc., pages 99–108, Aug. 1996.*
- C. Dyken and M. Reimers*, Real-time linear silhouette enhancement, Proceedings of Mathematical Methods for Curves and Surfaces 2004, July 2004, Tromso, Norway, pp. 135-143.*