Dissertação de Mestrado

Visualização de Rótulos em Objetos de Modelos Massivos em Tempo Real

Renato Deris Prado

Orientador: Alberto Barbosa Raposo

março 2013

Introdução

- Rótulos virtuais

  Informações textuais dispostas sobre superfícies geométricas.

- Armazenam diferentes informações.

  Nomes, numerações.

  Dados relevantes que precisem ser notados rapidamente.

Figura 1. Caixa com versão simples de rótulo virtual [Prado et al 2011].

Introdução

- Aplicações para rótulos

  Jogos.

  Visualizadores 3D.

    - Visualizadores de modelos CAD.

- Modelos CAD

  Visualização científica.

  Baseados em estruturas reais.

  Prédios, veículos, refinarias de petróleo.

  Desejável visualização imediata de informações.

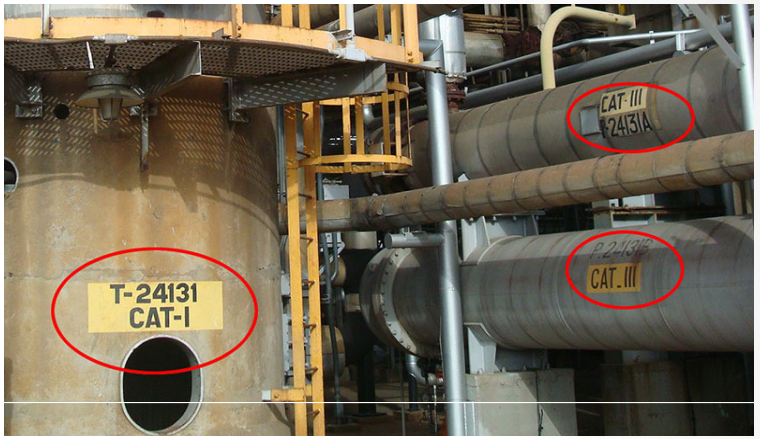


Figura 2. Foto de uma refinaria da Petrobras mostrando objetos com rótulos.

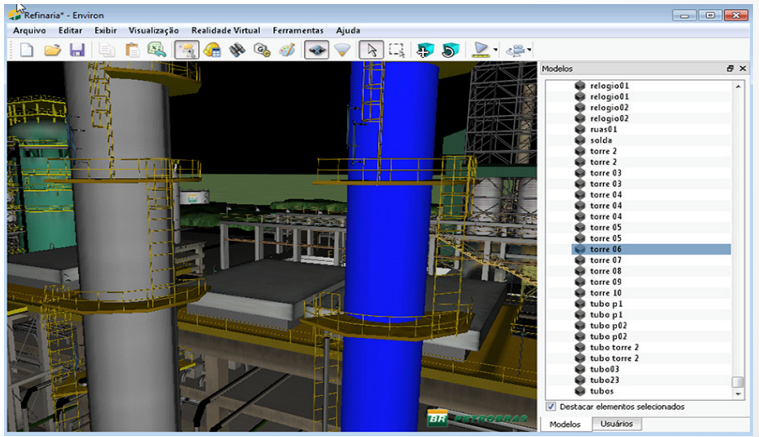


Figura 3. No Environ[ref], é preciso clicar em um objeto para saber seu nome

Introdução

- Modelos CAD

  Compostos por diferentes formas geométricas (Figura 4).

  Podem ser **massivos**, como refinaria contendo milhões ou bilhões de objetos (Figura 5).

  Rendering de modelos massivos pode apresentar alto custo computacional.

  Exibição de rótulos poderia agravar problemas de **memória** e **desempenho**.

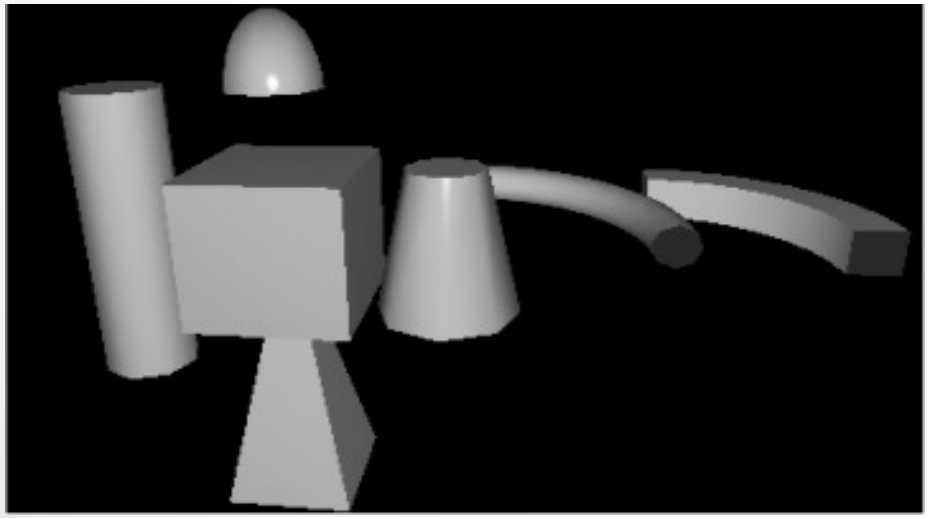


Figura 4. Exemplo de geometrias de modelos CAD

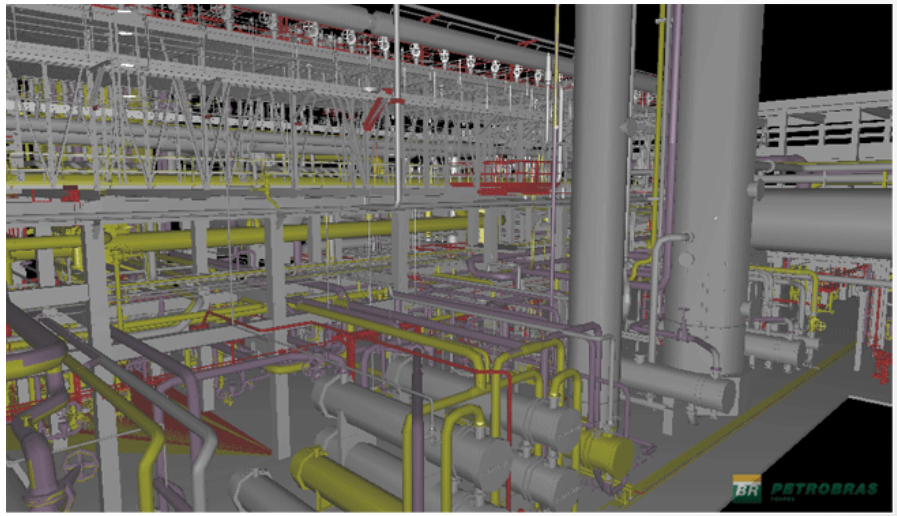


Figura 5. Refinaria de petróleo visualizada no software Environ [].

Introdução

- Desafios para a exibição de rótulos em modelos CAD

  Gastar o mínimo de **memória**.

  Não prejudicar o **desempenho.**

  Posicionar texto sobre diferentes geometrias.

  Lidar com problemas de ***aliasing***.

Que dificuldades eles podem apresentar?

Introdução

- Memória RAM e VRAM

  Modelos CAD podem ter milhões de objetos.

  Cada objeto possui um nome diferente.

  Se cada objeto possuísse uma textura diferente, a quantidade de memória gasta seria inviável.

  Necessidade de um gerenciamento de memória!

- Desempenho

  "Texturizar" todos os objetos visíveis geraria muitas trocas de contexto de textura por quadro.

  Muitas trocas de contexto de textura prejudicam o desempenho da aplicação [Prado et al.]

Introdução

- Posicionamento dos rótulos

  Objetos possuem formas e tamanhos diferentes.

  Como mapear o texto sobre diferentes superfícies e obter um bom efeito visual?

  De qualquer posição que objeto estiver sendo visualizado é preciso que seu rótulo esteja aparente.

  Orientação dos rótulos deve estar correta quando modelo é carregado.

- Aliasing

  O usuário pode se aproximar ou se afastar de um objeto. Isso pode causar *aliasing* em seu rótulos.

  Esses problemas são conhecidos como “texture magnification” e “texture minification”.

Trabalhos relacionados

- Aveva Review [Aveva Review 2010]

  Exibe *tags* em cilindros e caixas apenas.

  Limita número de objetos que recebem *tags* por quadro.

  Implementação não conhecida.

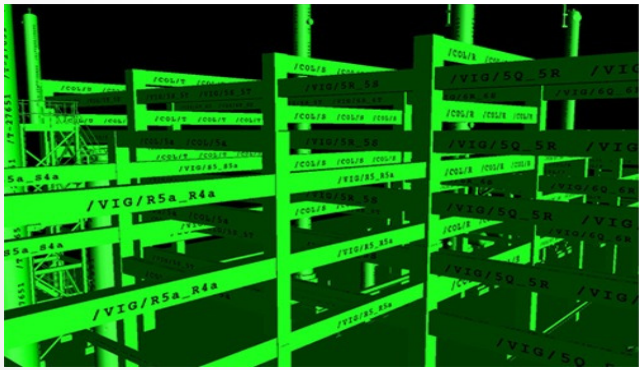


Figura 6. *Screenshot* do software Aveva Review

- Autotag [Prado et al. 2011]

  Cada objeto possui uma textura (cilindros e caixas apenas).

  Não há gerenciamento de memória.

  Limita objetos que recebem *tags* por quadro.

Figura 7. Objetos com tags. [Prado et al. 2011]

Trabalhos relacionados

- Texture Sprites [Lefebvre et al. 2005]

  Pequenas texturas (*sprites*) que podem formar uma maior.

  Utiliza estrutura similar a *octree* que pode ser mapeada em uma textura 3D em GPU.

  Octree guarda informações de cada *sprite* como posição, tamanho e id da textura.

Figura 8. "Bunnys" com sprites [Lefebvre et al.]

- Real-Time Texture-Mapped Vector Glyphs [Qin et al. 2006]

  Texto é representado de forma vetorial para reduzir aliasing quando magnificado.

  Utiliza estratégia similar a de Texture Sprites para posicionar palavras sobre objetos 3D.

Figura 9. "Bandeira com texto [Qin et al.]

Trabalhos relacionados

- Text Scaffolds [Cipriano et al. 2008]

  Posicionamento de texto em superfícies de curvatura acentuada e até com buracos.

  Utiliza andaimes para guardar as letras.

  Posiciona as letras flutuando sobre os objetos.

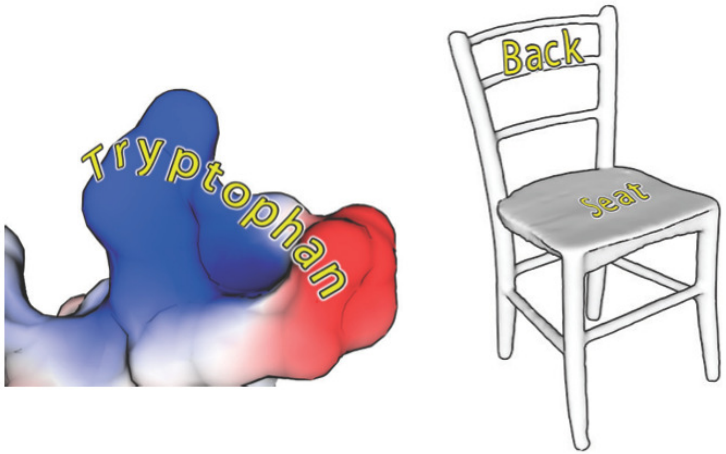


Figura 10. Objetos com *scaffolds* [Cipriano et al.]

Trabalhos relacionados

- Conclusão

  Poucos trabalhos abordam o problema de exibir texto em modelos massivos.

  Os que abordam apresentam algumas limitações.

  - Limite de objetos com rótulos e texto em caixas e cilindros apenas.

- Proposta deste trabalho

  Desenvolver técnica para exibição de rótulos em modelos massivos.

  Atacar os problemas de desempenho, memória, posicionamento e *aliasing*.

  Independente do grande número de objetos desempenho não é prejudicado.

Rótulos em modelos massivos

1. Aplicação

Carrega objetos e cria informações textuais.

Cria textura de atlas com caracteres.

Informações textuais (códigos ASCII)

B O X 0 1	B O X 0 2	B O X 0 3
B O X 0 4	B O X 0 5	B O X 0 6
B O X 0 7	O B J 5 0	O B J 4 3

Imagem com caracteres

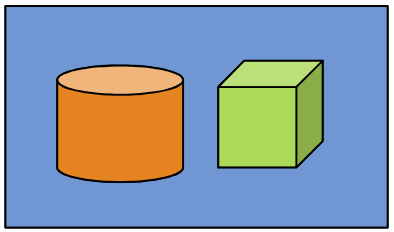
A	B	C	D	E	F	G
H	I	J	K	L	M	N
P	Q	R	S	T	U	V
X	Z	...				

2. Primeira passada

Gera coordenadas de textura para os objetos

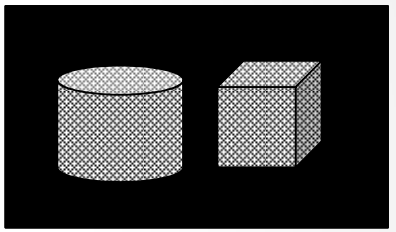
Escreve em dois *buffers* fora da tela

Cores



fragData = vec(r, g, b, a)

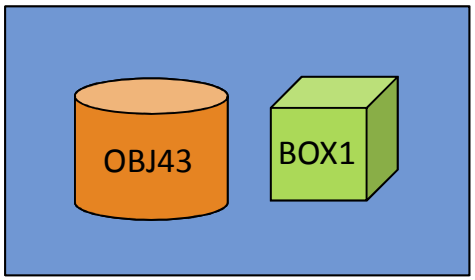
Informações de textura



fragData = vec(s, t, id, n_{chars})

3. Segunda passada

Consulta todas essas informações e gera cena final com rótulos.



1. Aplicação

Carrega objetos e cria informações textuais.

Cria textura de atlas com caracteres.

Informações textuais (códigos ASCII)

BOX01	BOX02	BOX03
BOX04	BOX05	BOX06
BOX07	OBJ50	OBJ43

Imagem com caracteres

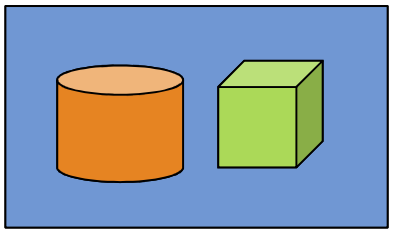
A	B	C	D	E	F	G
H	I	J	K	L	M	N
P	Q	R	S	T	U	V
X	Z	...				

2. Primeira passada

Gera coordenadas de textura para os objetos

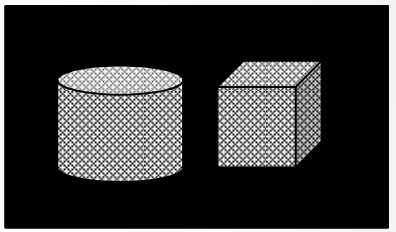
Escreve em dois *buffers* fora da tela

Cores



fragData = vec(r, g, b, a)

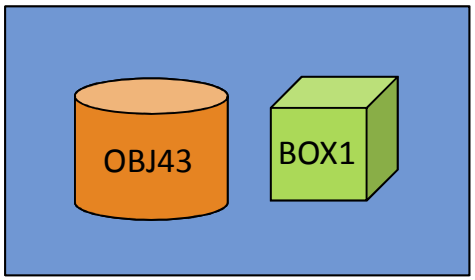
Informações de textura



fragData = vec(s, t, id, n_{chars})

3. Segunda passada

Consulta todas essas informações e gera cena final com rótulos.



Rótulos em modelos massivos

- Armazenamento de informações textuais dos rótulos.

  Utilizada textura 2D.

  Cada *texel* contém 4 bytes e pode armazenar 4 caracteres ASCII.

  Cada rótulo possui um identificador (valor em verde na Figura 11).

  Cada objeto pode ser associado a um identificador de rótulo.

...															
/	O	B	J		4	5	6	B	O	X	1				
5								6							
		Q	U	A	D			O	B	J	E	C	T	-	2
3								4							
C	Y	L	I	N	D	E	R	O	B	J	4	3			
1								2							

2 *texels* por rótulo
2 rótulos por linha

Figura 11. Armazenamento de códigos ASCII de todos os rótulos.

Rótulos em modelos massivos

- Atlas com todos os caracteres

  Caracteres foram organizados seguindo a ordem da codificação ASCII (Figura 12).

  Assim, é simples encontrar suas posições (equações 3-1 a 3-3):

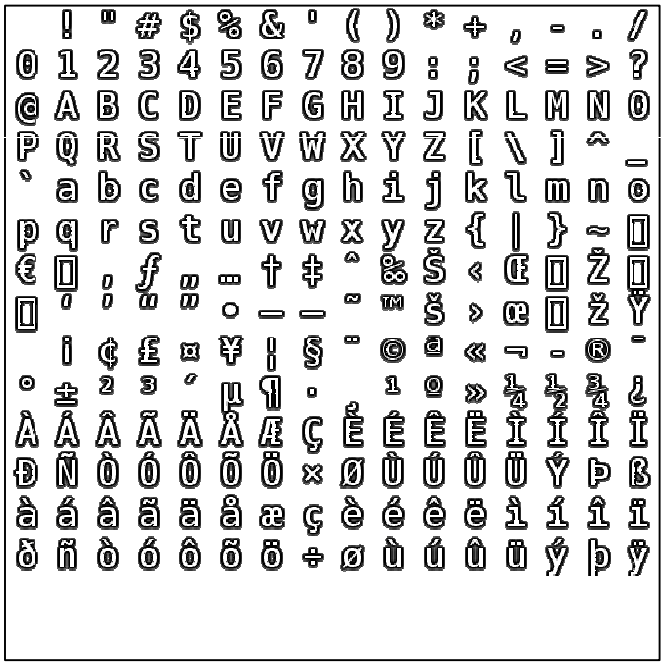
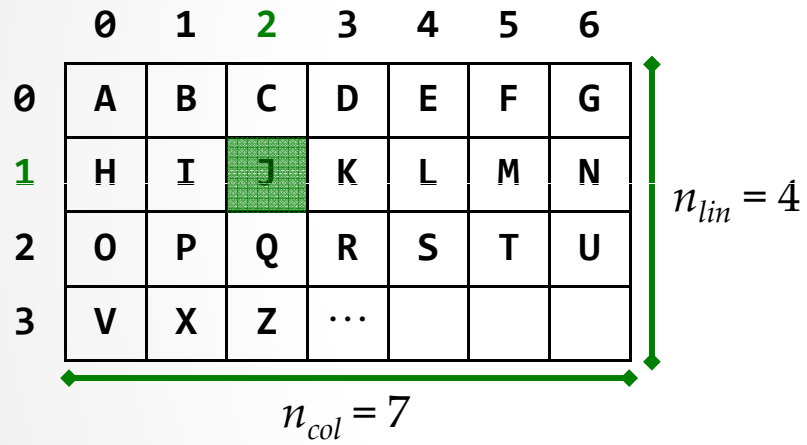


Figura 12. Atlas com caracteres organizados na ordem da codificação ASCII.

Rótulos em modelos massivos

- Atlas com todos os caracteres

  Exemplo de como encontrar posição de um caractere:



$$i = c_{des} - c_{firstChar} = c - 32 \qquad (3-1)$$

$$col = i \bmod n_{col} \qquad (3-2)$$

$$lin = floor\left(i / n_{col} \right) \qquad (3-3)$$

$c_{des} = c_j = 74$

$c_{firstChar} = c_a = 65$

$i = 74 - 65 = 9$

i: índice do caractere procurado.

c_{des}: código ASCII do caractere procurado.

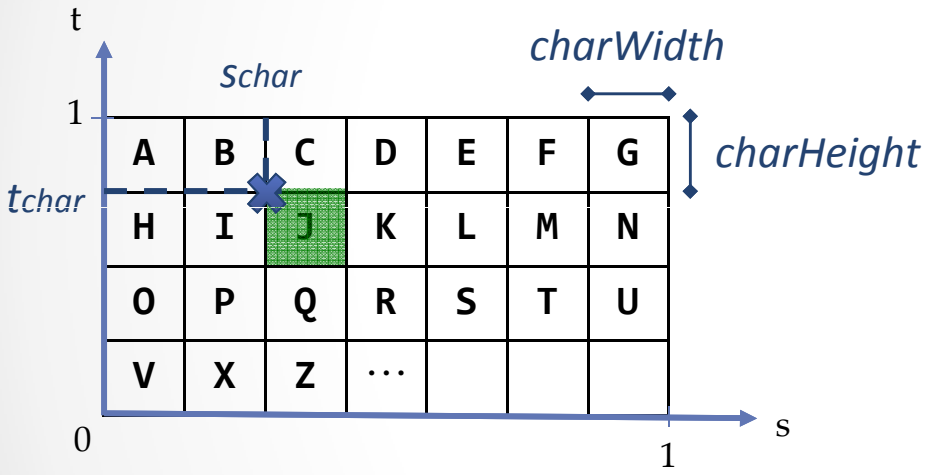
$c_{firstchar}$: código ASCII do caractere inicial.

col e ***lin***: coluna e linha resultantes.

Rótulos em modelos massivos

- Atlas com todos os caracteres

  Posição do caractere no espaço da textura:



charHeight e charWidth iguais para todos os chars

$$charWidth_{atlas} = 1/n_{col} \quad (3-4)$$

$$charHeight_{atlas} = 1/n_{lin} \quad (3-5)$$

$$s_{char} = col * charWidth_{atlas} \quad (3-6)$$

$$t_{char} = 1-(lin * charHeight_{atlas}) \quad (3-7)$$

ncol e nlin: colunas e linhas de caracteres do atlas.

col e ***lin***: coluna e linha das equações (3-2) e (3-3)

s_{char} e t_{char}: posição do caractere no espaço da textura.

1. Aplicação

Carrega objetos e cria informações textuais.

Cria textura de atlas com caracteres.

Informações textuais (códigos ASCII)

BOX01	BOX02	BOX03
BOX04	BOX05	BOX06
BOX07	OBJ50	OBJ43

Imagem com caracteres

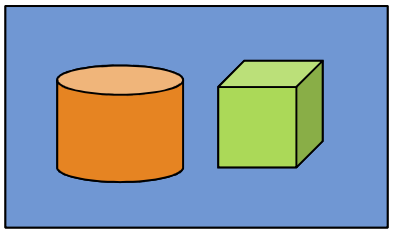
A	B	C	D	E	F	G
H	I	J	K	L	M	N
P	Q	R	S	T	U	V
X	Z	...				

2. Primeira passada

Gera coordenadas de textura para os objetos

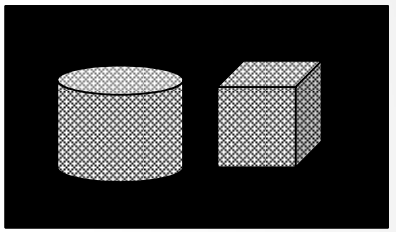
Escreve em dois *buffers* fora da tela

Cores



fragData = vec(r, g, b, a)

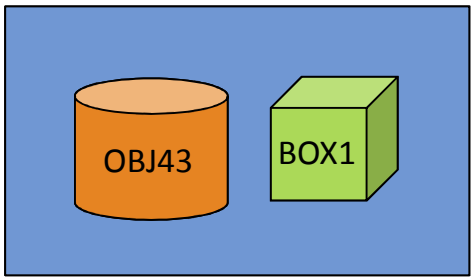
Informações de textura



fragData = vec(s, t, id, n_{chars})

3. Segunda passada

Consulta todas essas informações e gera cena final com rótulos.



Rótulos em modelos massivos

- Primeira passada de *rendering*

  Conhece identificador de rótulo e seu tamanho, para cada objeto.

  No *vertex shader* são geradas as coordenadas de textura dos objetos.

  O fragment shader escreve em dois *buffers* (texturas) fora da tela (Figura 13).

  A segunda passada irá utilizar esses informações.

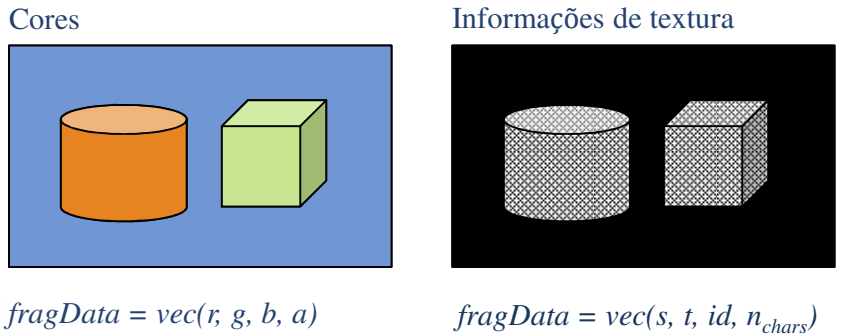


Figura 13. *Buffers* preenchidos na primeira passada de *rendering*.

1. Aplicação

Carrega objetos e cria informações textuais.

Cria textura de atlas com caracteres.

Informações textuais (códigos ASCII)

BOX01	BOX02	BOX03
BOX04	BOX05	BOX06
BOX07	OBJ50	OBJ43

Imagem com caracteres

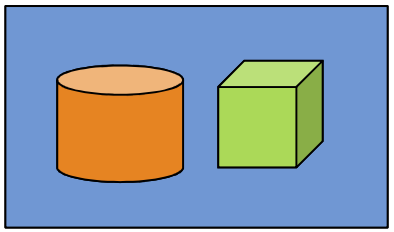
A	B	C	D	E	F	G
H	I	J	K	L	M	N
P	Q	R	S	T	U	V
X	Z	...				

2. Primeira passada

Gera coordenadas de textura para os objetos

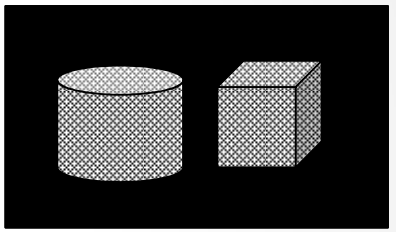
Escreve em dois *buffers* fora da tela

Cores



fragData = vec(r, g, b, a)

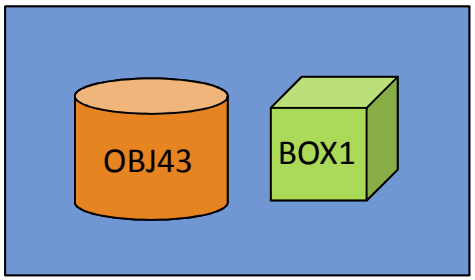
Informações de textura



fragData = vec(s, t, id, n_{chars})

3. Segunda passada

Consulta todas essas informações e gera cena final com rótulos.



Rótulos em modelos massivos

- Segunda passada de *rendering*

  Utiliza *Frame Buffer* padrão.

  Desenha *quad* do tamanho da tela e define suas coordenadas de textura, Figura 14 (a).

  Obtém informações de cor e de textura para cada fragmento do *quad*, Figura 14 (b).

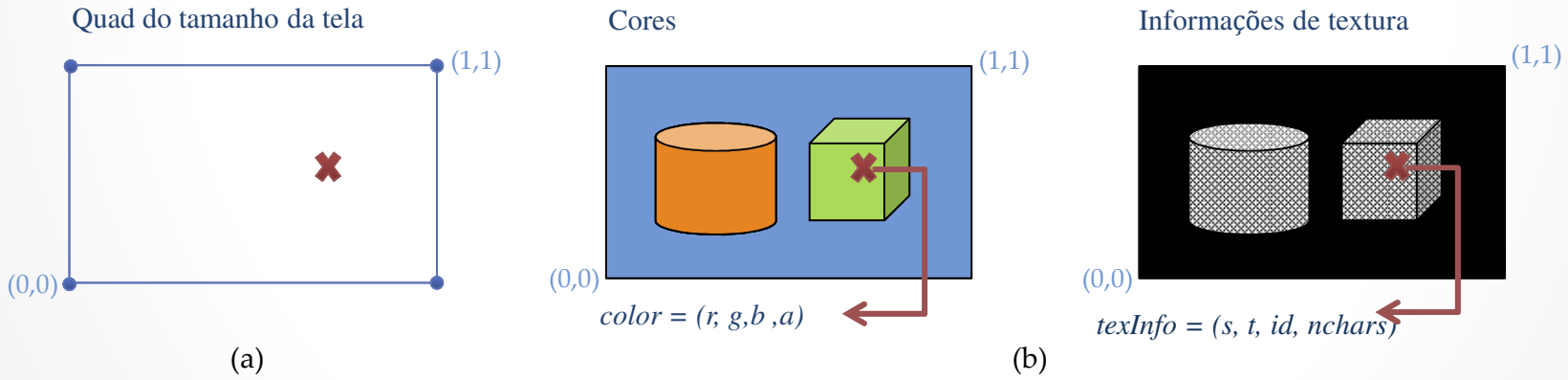


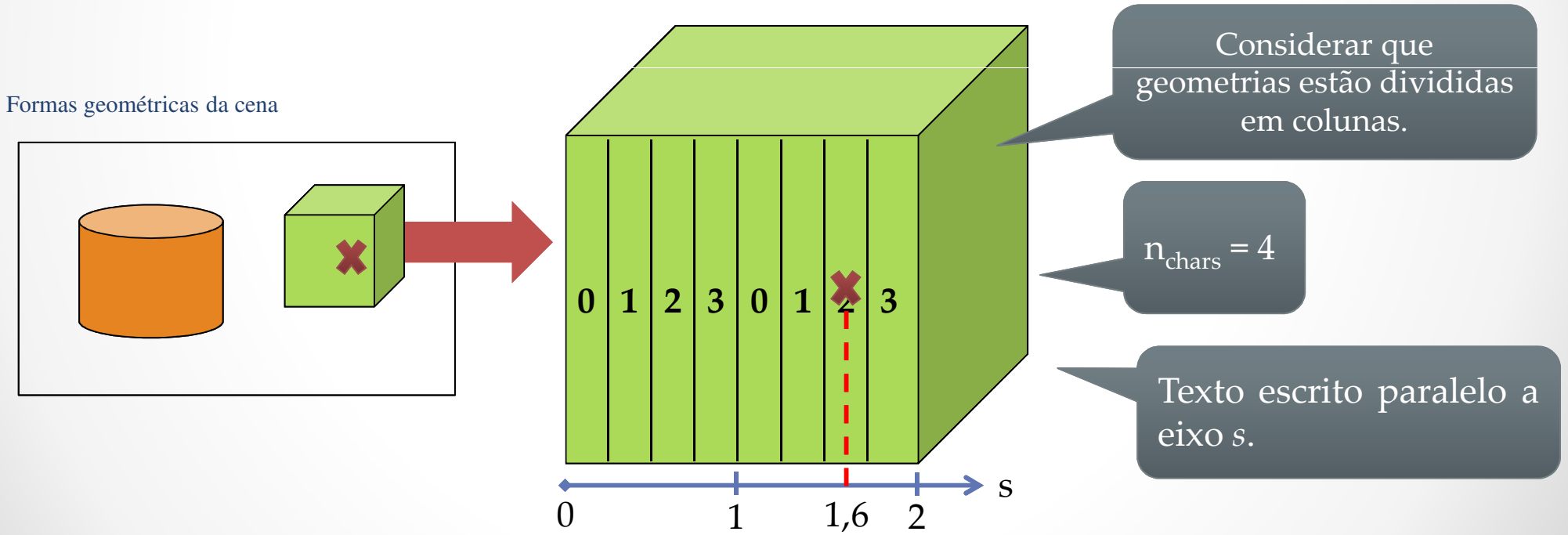
Figura 14. *Quad* do tamanho da tela (a) e amostragem dos *buffers* da primeira passada (b).

Rótulos em modelos massivos

- Segunda passada de *rendering*

  Quad do tamanho da tela possui nova cor e novas coordenadas de textura.

  Etapa 1: Para coordenada *s*, obter índice do caractere dentro do rótulo (não o código ASCII).

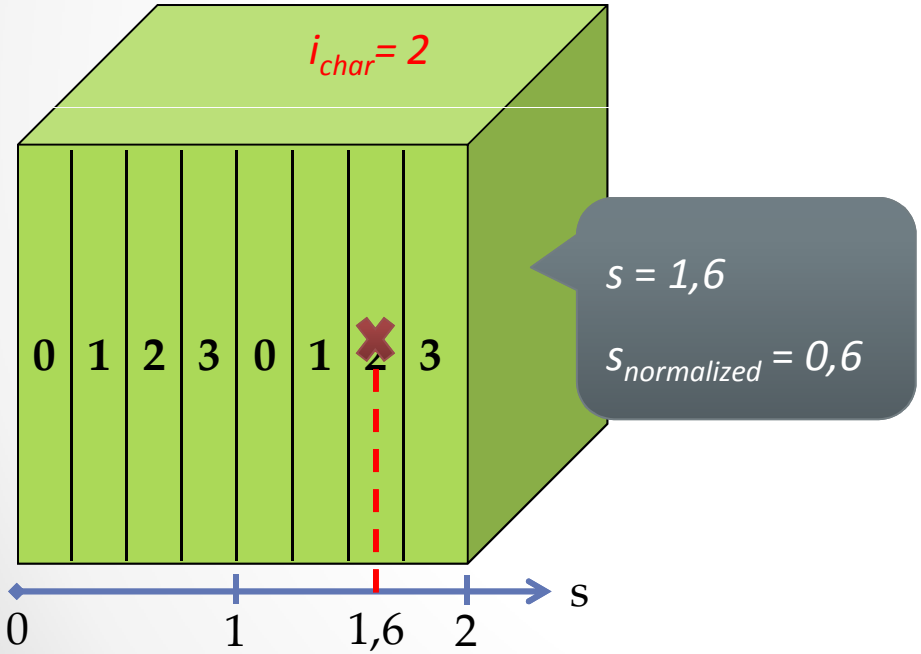


Rótulos em modelos massivos

- Segunda passada de *rendering*

  Quad do tamanho da tela possui nova cor e novas coordenadas de textura.

  Etapa 1: Para coordenada *s*, obter índice do caractere dentro do rótulo (não o código ASCII).



$$s_{normalized} = s - floor(s) \quad (3-8)$$

$$t_{normalized} = t - floor(t) \quad (3-9)$$

$$i_{char} = floor(s_{normalized} * n_{chars}) \quad (3-10)$$

$s_{normalized}$ e $t_{normalized}$: coordenadas de textura normalizadas para intervalo 0 a 1.

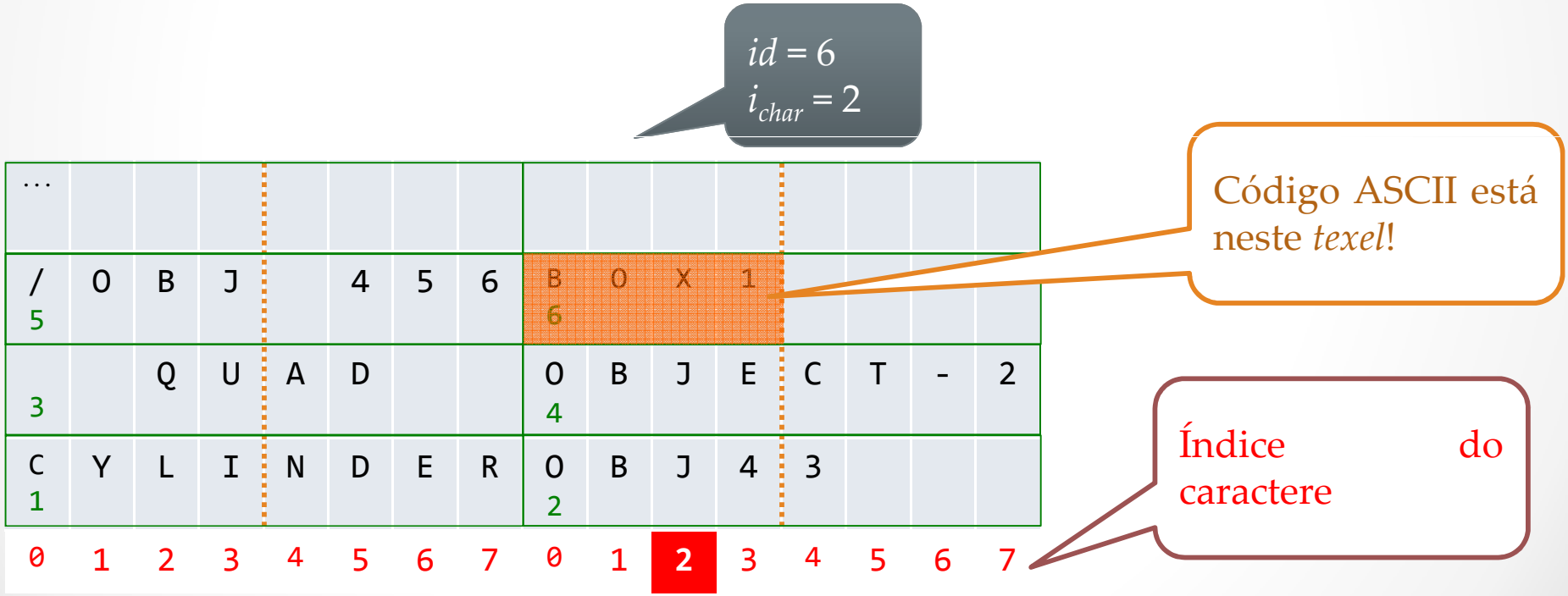
i_{char}: índice do caractere dentro do rótulo

Rótulos em modelos massivos

- Segunda passada de *rendering*

Etapa 2: Descobrir o código ASCII do caractere no atlas de informações textuais.

Já conhecemos o *id* e i_{char}, associados ao fragmento corrente!



Rótulos em modelos massivos

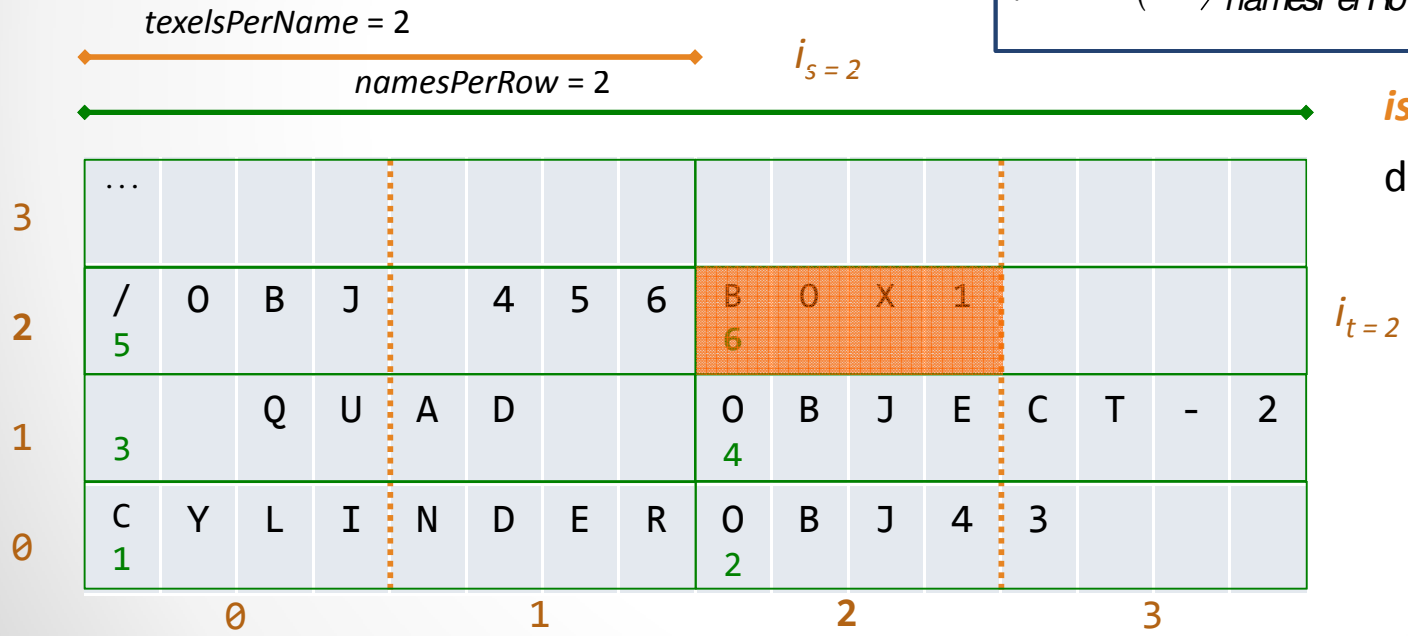
- Segunda passada de *rendering*

Etapa 2: Descobrir o código ASCII do caractere no atlas de informações textuais.

Como encontrar *texel*?

$$i_s = ((id-1) \bmod namesPerRow) * texelsPerName + floor(i_{char} / 4) \quad (3-11)$$

$$i_t = floor\left(id-1 / namesPerRow\right) \quad (3-12)$$



is e it: *índices horizontal e vertical* do *texel* procurado.

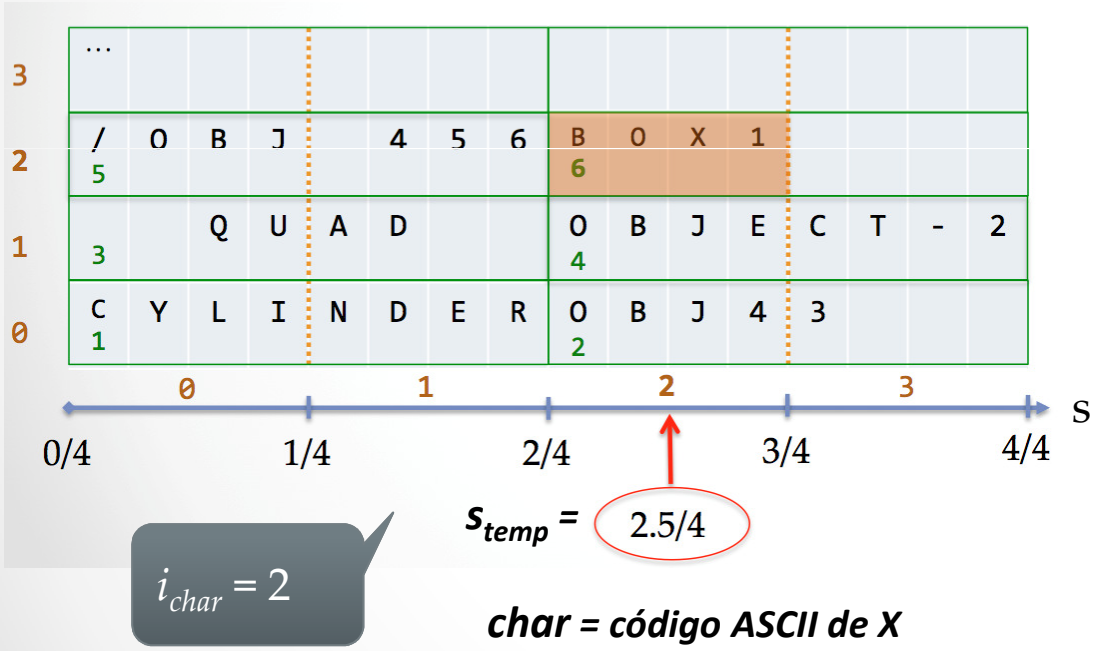
$id = 6$
$i_{char} = 2$

Rótulos em modelos massivos

- Segunda passada de *rendering*

Etapa 2: Descobrir o código ASCII do caractere no atlas de informações textuais.

Coordenadas de textura do *texel*?



$$s_{temp} = (i_s + 0.5)/w \qquad (3-13)$$

$$t_{temp} = (i_t + 0.5)/h \qquad (3-14)$$

$$char = texel[i_{char} \bmod 4] \qquad (3-15)$$

s_{temp} e t_{temp}: *coordenadas de textura do texel procurado*

i_{char}: *índice do caractere no rótulo.*

char: *código ASCII do caractere procurado*

Rótulos em modelos massivos

- Segunda passada de *rendering*

Etapa 3: Amostrar atlas de caracteres na posição correta.

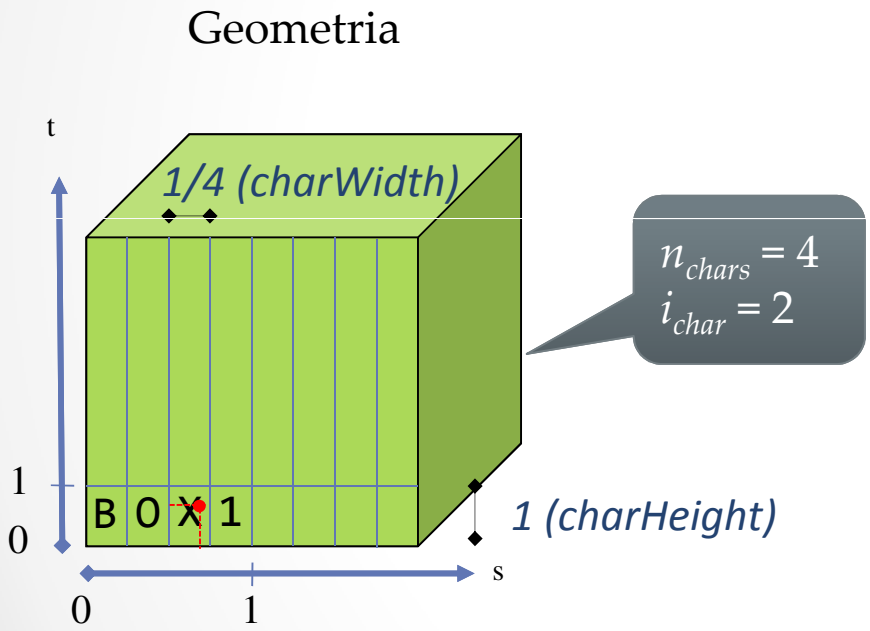


Figura 19. Um fragmento e seu offset em relação ao início de uma coluna e de uma linha

$$charWidth_{geom} = 1/n_{chars} \quad (3-16)$$

$$sOffset_{geom} = s_{normalized} - (i_{char} * charWidth_{geom}) \quad (3-17)$$

$$tOffset_{geom} = t_{normalized} \quad (3-18)$$

$charWidth_{geom}$ e $charHeight_{geom}$: largura e altura do caractere no espaço de textura.

$s_{normalized}$ e $t_{normalized}$: encontrados na Etapa 1.

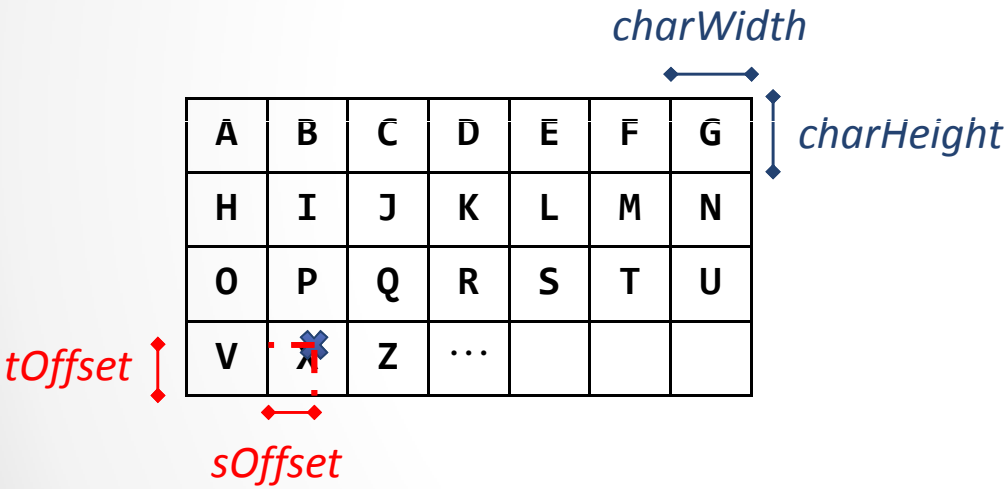
i_{char}: índice do caractere dentro do rótulo.

t_{Offset} e s_{Offset}: *offset* do fragmento em relação ao início de uma linha e de uma coluna.

Rótulos em modelos massivos

- Segunda passada de *rendering*

Etapa 3: Descobrir a posição correta para o fragmento, no atlas de caracteres



$$r = charWidth_{atlas} / charWidth_{geom} \quad (3-19)$$

$$sOffset_{atlas} = sOffset_{geom} * r \quad (3-20)$$

$$tOffset_{atlas} = tOffset_{geom} * charHeight_{atlas} \quad (3-21)$$

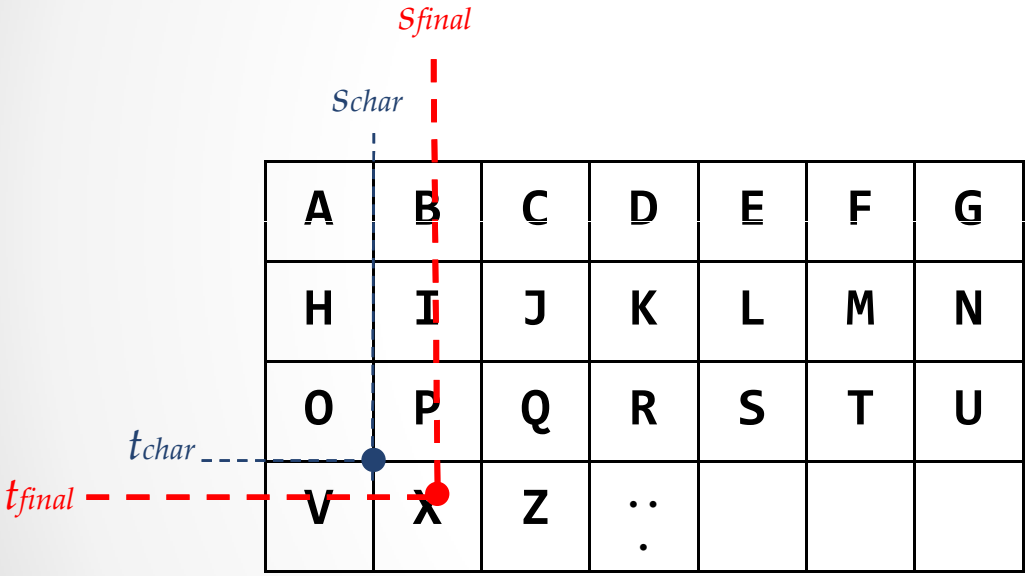
r: razão entre largura do caractere na geometria e no atlas.

$sOffset_{atlas}$ e **$tOffset_{atlas}$**: *offset* no atlas relacionado ao *offset* do fragmento na geometria.

Rótulos em modelos massivos

- Segunda passada de *rendering*

Etapa 3: Descobrir a posição correta para o fragmento, no atlas de caracteres



$$s_{final} = s_{char} + sOffset_{atlas} \qquad (3-22)$$

$$t_{final} = t_{char} - tOffset_{atlas} \qquad (3-23)$$

***Schar* e *tchar*:** posição inicial do caractere no espaço da textura, calculados com equações (3-1) a (3-7)

sfinal e ***tfinal***: coordenadas para amostrar o atlas de caracteres.

Posicionamento dos rótulos

- Geração de coordenadas de textura

  As coordenadas de textura dos objetos devem ser geradas na primeira passada.

  Segunda passada espera aceita repetição de texto.

  Centralizar texto: identificador de rótulos igual a 0 para alguns fragmentos, Figura 18 (b) e Figura 18(c).

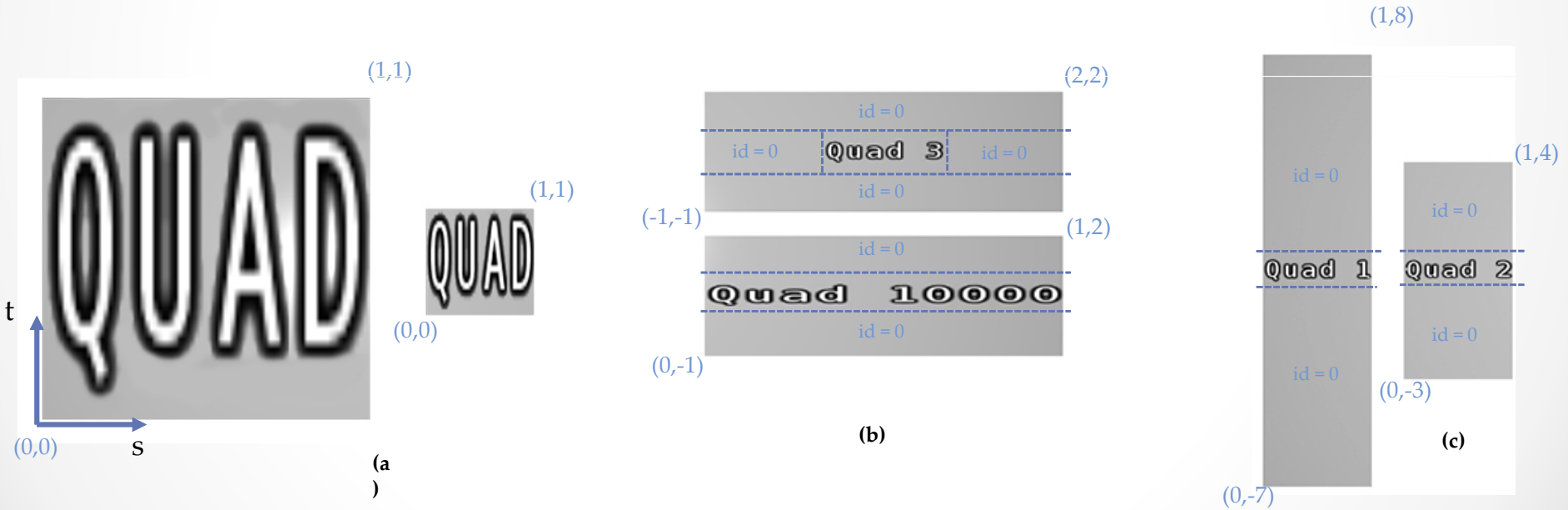


Figura 20. Rótulos esticam dependendo das dimensões do *quad* (a); posicionamento horizontal considera o a quantidade de caracteres (b); posicionamento vertical de rótulos (c).

Posicionamento dos rótulos

- Geração de coordenadas de textura
  - "Aspect ratio" deve ser considerado.
  - Espaços para separar os rótulos horizontalmente.
  - Repetição horizontal e centralização vertical.

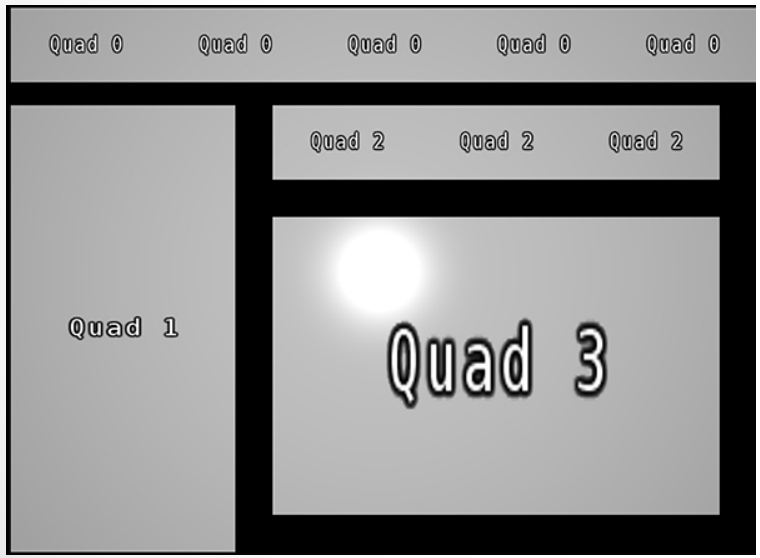


Figura 21. Quads de tamanhos variados e posicionamento automático de rótulos

- Cilindros, *dishes*, esferas e caixas
  - Cálculo baseado nos *quads*.
  - Eixos foram invertidos para os cilindros.

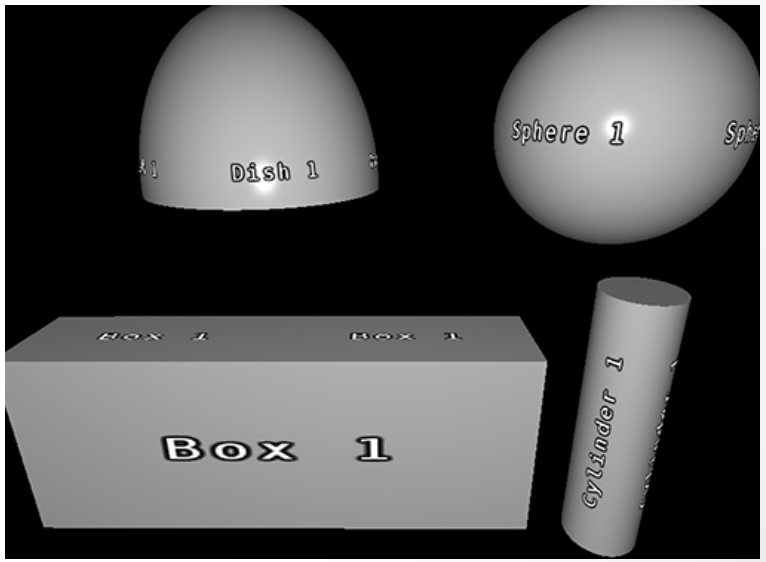


Figura 22. Diferentes primitivas com rótulos.

Rótulos em modelos reais

- *Engine* gráfica para modelos CAD reais

  Maioria dos objetos são cilindros e caixas.

  Problema: *Aliasing*

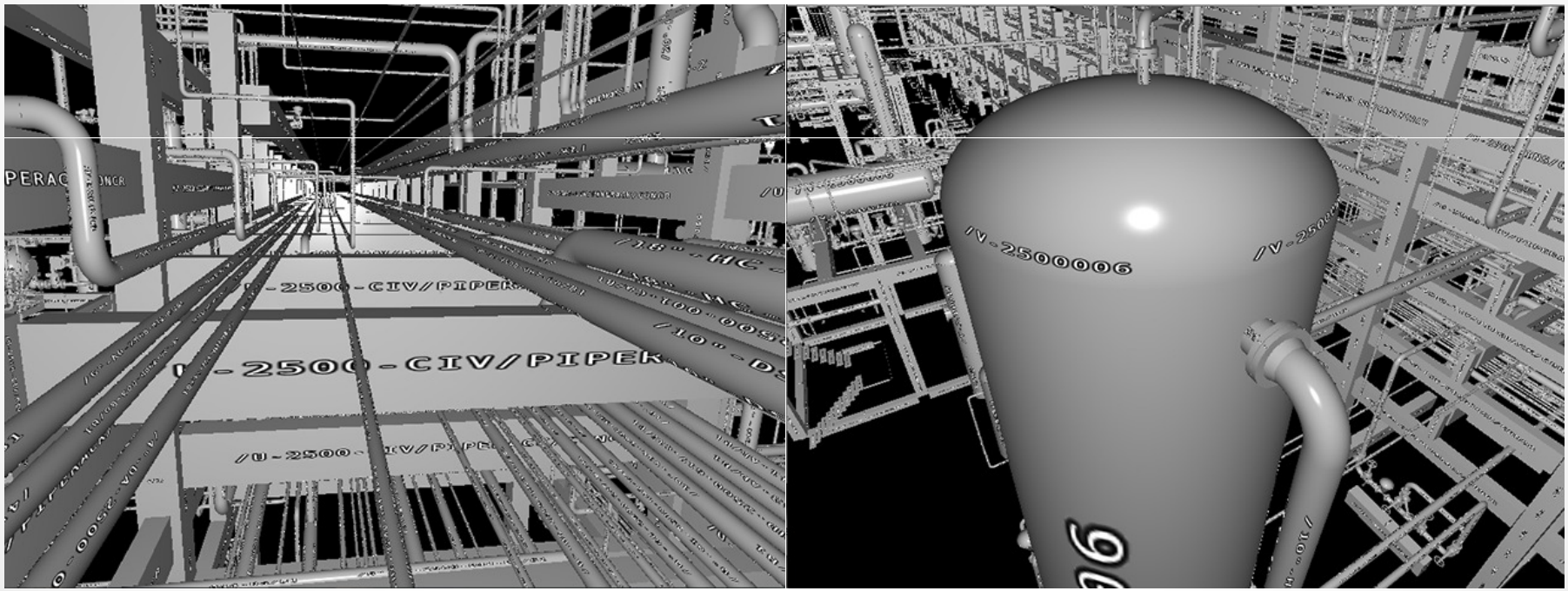


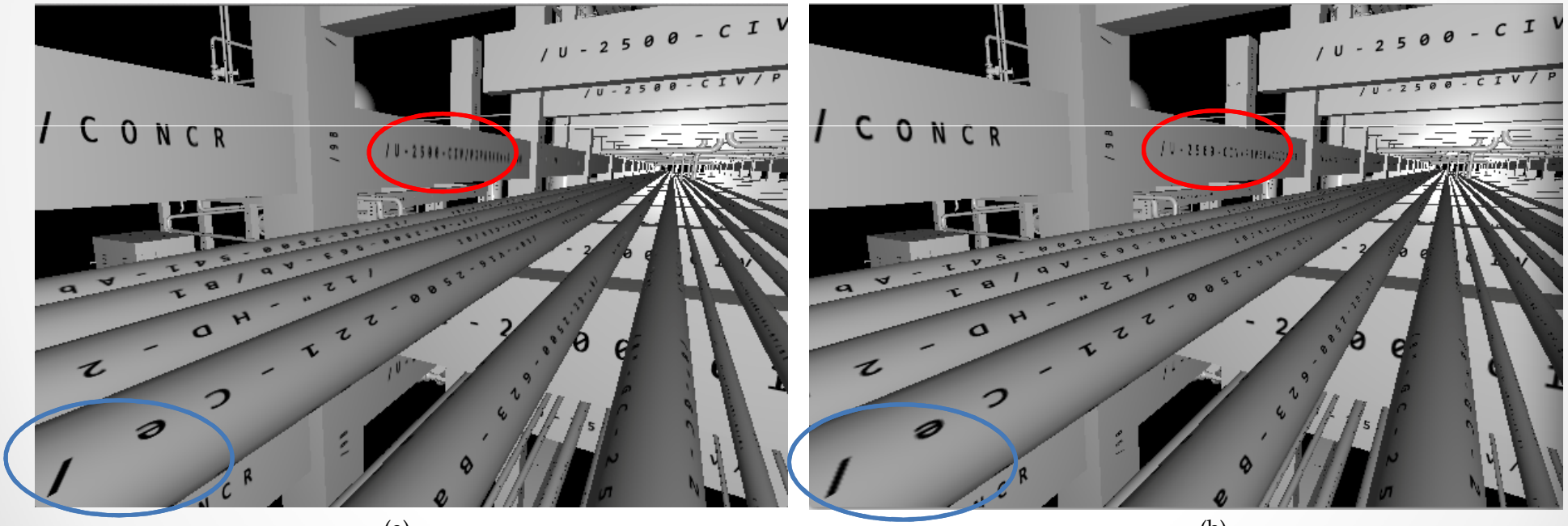
Figura 23. *Screenshots* de rótulos em modelo CAD

Antialising e qualidade visual

- *Mipmapping*

  Possibilidade de melhorar o problema de ampliação de textura, sem piorar a redução.

  Os atlas a partir de 32 x 32 *pixels* não representam bem os caracteres.



(a) (b)

Figura 24. Comparação de cena exibindo modelo CAD, com (a) e sem (b) o uso de *mipmapping*. O círculo em vermelho destaca pixels afastados da tela e o círculo em azul, pixels bem próximos.

Antialising e qualidade visual

- Cores

  Utilizado um atlas de caracteres sem bordas e com letras pretas.

  No *fragment shader* da segunda passada é possível modificar essa cor.

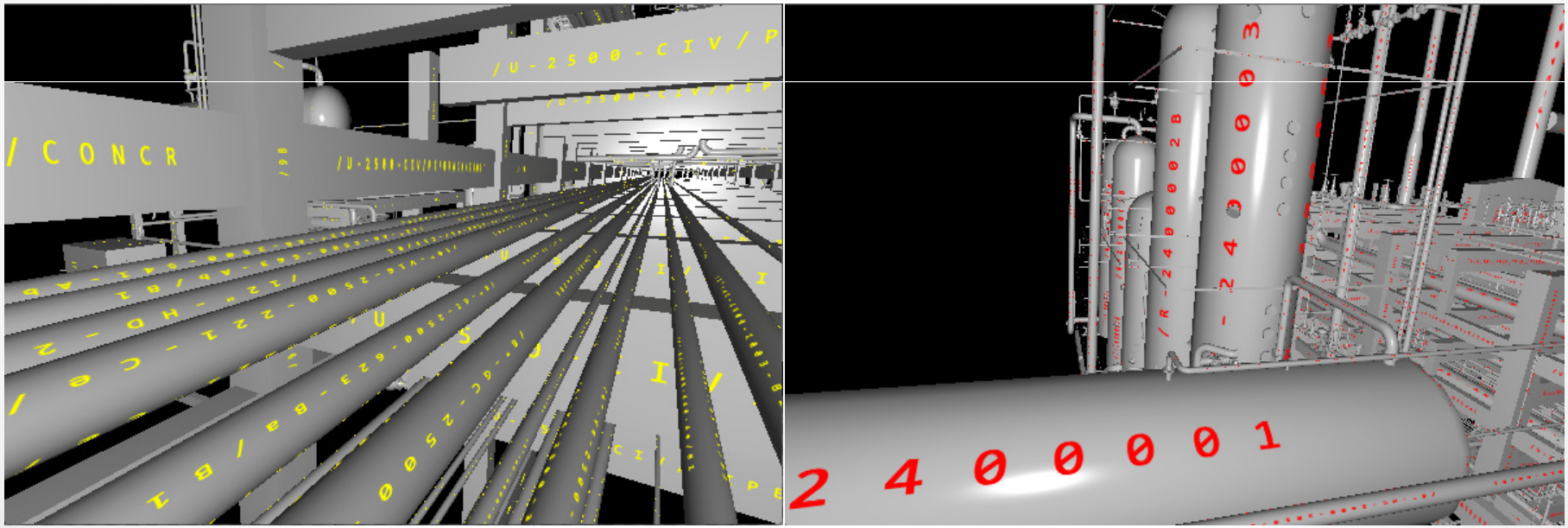


Figura 25. *Screenshots* de rótulos com cores em modelos CAD

Resultados de desempenho

- Teste 1 – *Quads* e posição única de câmera

  A Figura *** mostra como um exemplo de como a cena foi organizada.

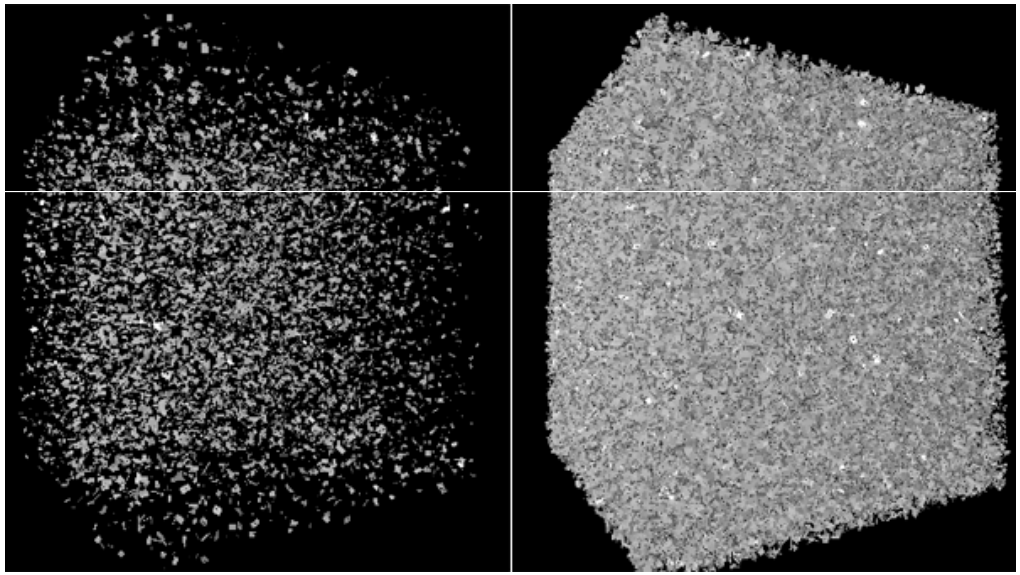


Figura 25. Organização da cena para testes

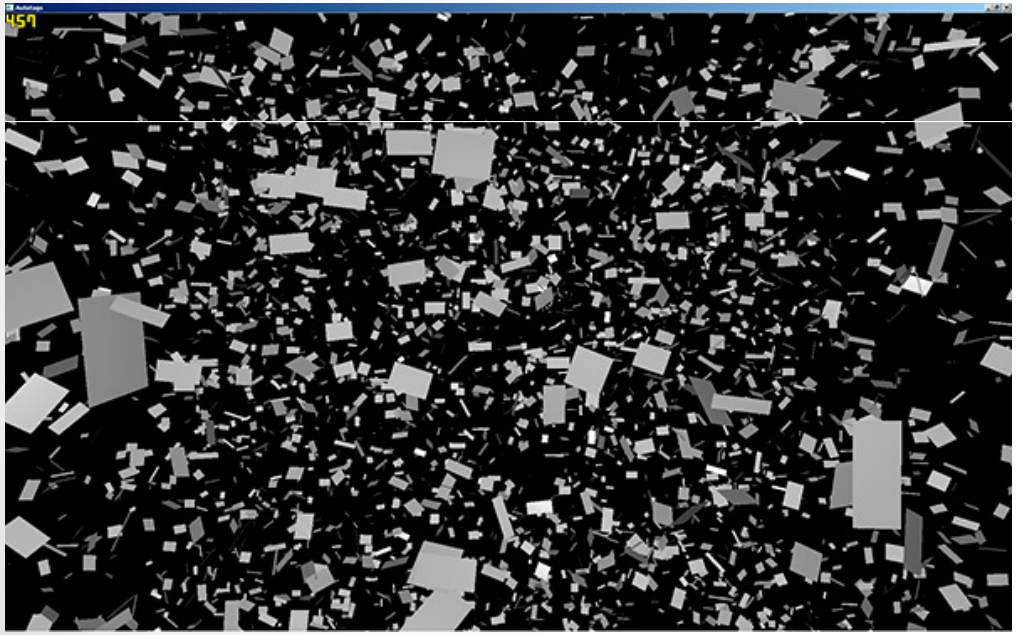
Máquina: Windows 7 64 bits, processador Intel® Core I7™ 870 de 2.93 GHz, 8GB de memória RAM e placa gráfica GeForce GTX 580.

Resultados de desempenho

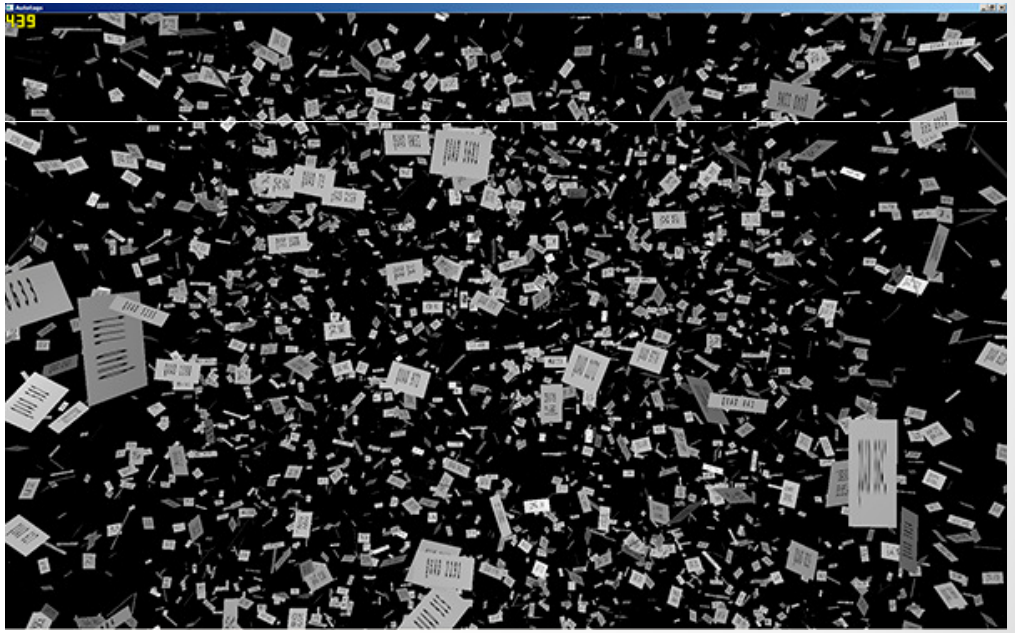
- Teste 1 – *Quads* e posição fixa de câmera

  10.000 objetos (1920x1200)

SEM RÓTULOS – 457 FPS



COM RÓTULOS – 439 FPS

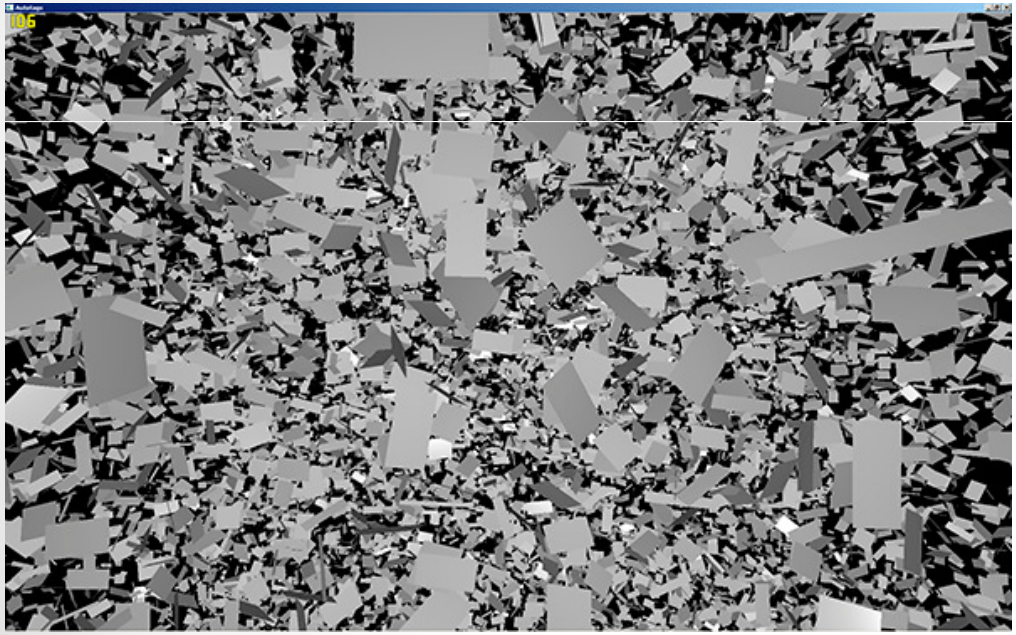


Resultados de desempenho

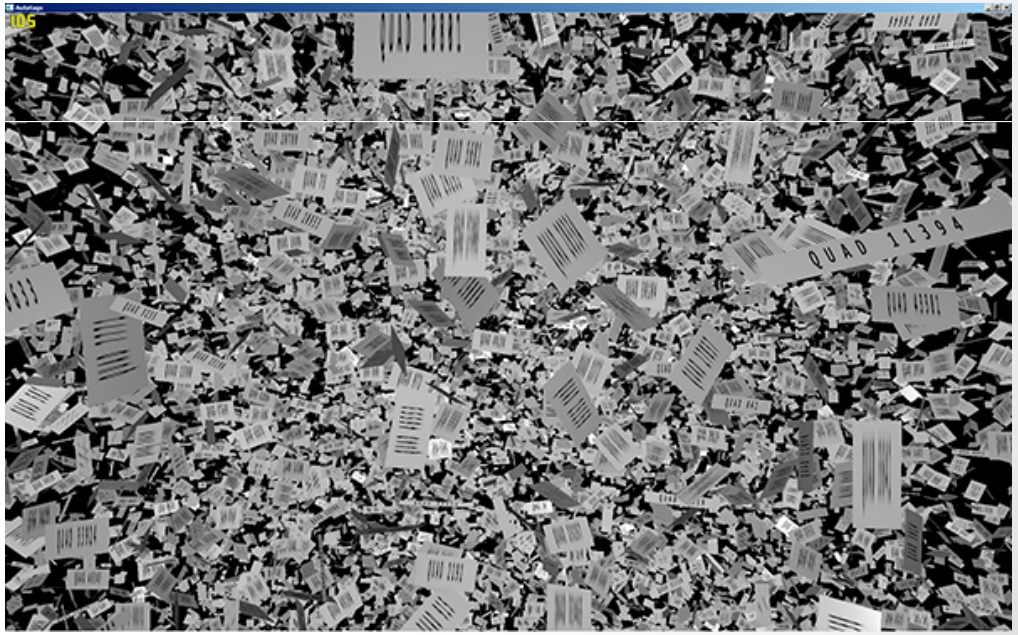
- Teste 1 – *Quads* e posição fixa de câmera

  50.000 objetos (1920x1200)

SEM RÓTULOS – 106 FPS



COM RÓTULOS – 105 FPS

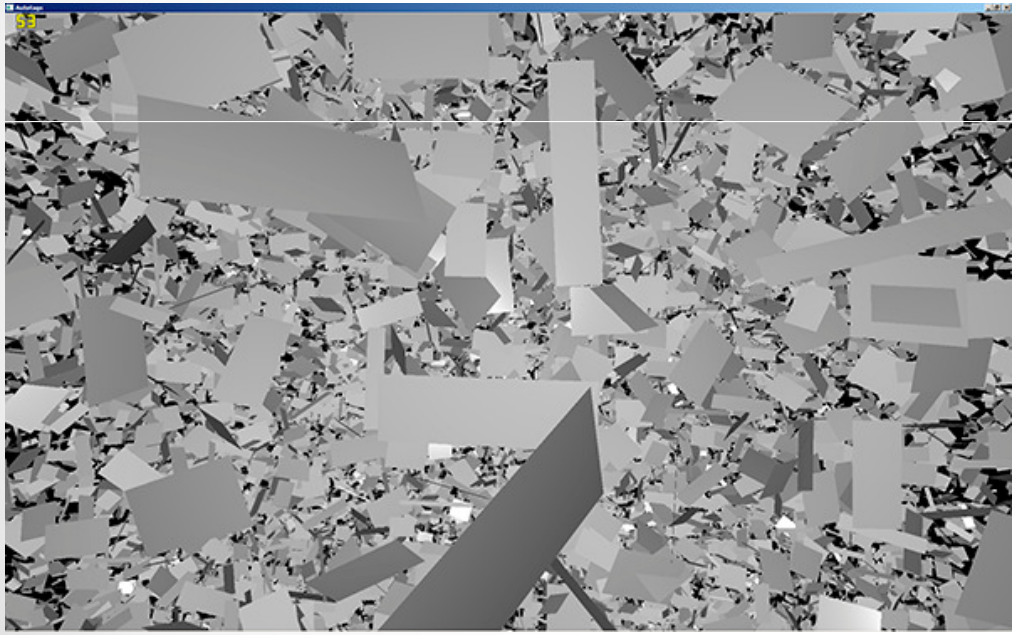


Resultados de desempenho

- Teste 1 – *Quads* e posição fixa de câmera

  100.000 objetos (1920x1200)

SEM RÓTULOS – 53 FPS



COM RÓTULOS – 54 FPS

Resultados de desempenho

- Teste 1 – *Quads* e posição fixa de câmera

Objetos	Resolução	FPS (Sem Rótulos)	FPS (Com Rótulos)	Diferença
200.000	1920 x 1200	27	27	0
500.000	1920 x 1200	11	11	0
1.000.000	1920 x 1200	6	6	0

Resultados de desempenho

- Teste 2 – *Quads* e navegação pela cena

  Poucos objetos.

  Média de FPS.

Objetos	Resolução	FPS (Sem Rótulos)	FPS (Com Rótulos)	Diferença
10.000	600 x 400	511,6	485,3	26,3
	1920 x 1200	488,1	470,1	9,0
50.000	600 x 400	106,7	107,9	-1,2
	1920 x 1200	102,0	99,9	2,1

Resultados de desempenho

- Teste 2 – *Quads* e navegação pela cena

  Muitos objetos.

  Média de FPS.

Objetos	Resolução	FPS (Sem Rótulos)	FPS (Com Rótulos)	Diferença
500.000	600 x 400	10,6	10,6	0
	1920 x 1200	10,6	10,5	0,1
1.000.000	600 x 400	4,5	5,3	-0,8
	1920 x 1200	5,4	5,4	0
1.200.000	600 x 400	4,6	4,5	0,1
	1920 x 1200	4,5	4,5	0

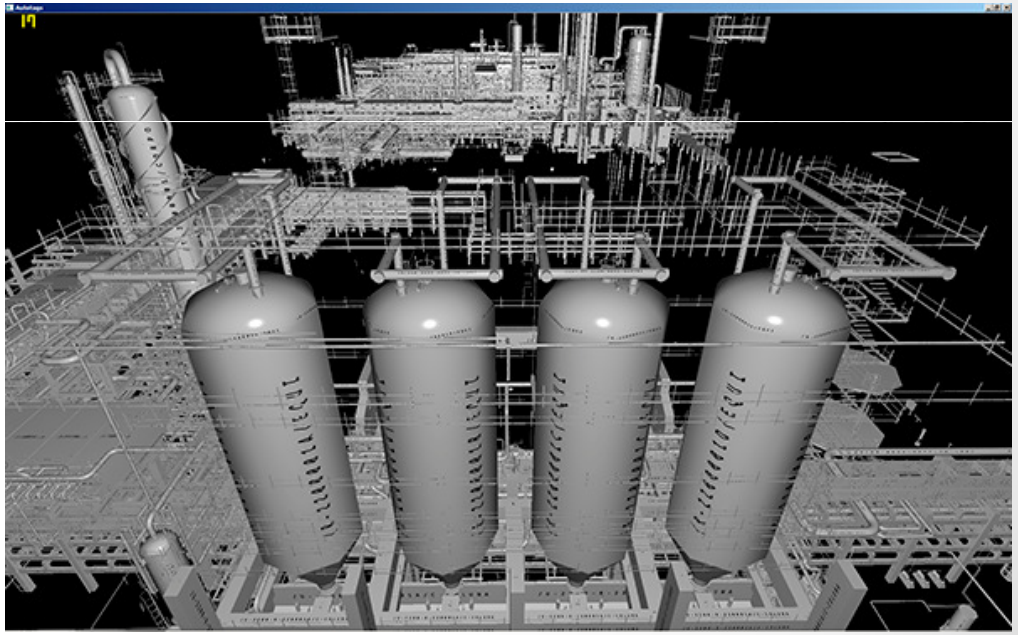
Resultados de desempenho

- Testes 3 - CAD e posição única de câmera

  300.000 objetos

SEM RÓTULOS – 17 FPS

COM RÓTULOS – 17 FPS



Resultados de desempenho

- Testes 4 - CAD e navegação pela cena

Objetos	Resolução	FPS (Sem Rótulos)	FPS (Com Rótulos)	Diferença
300.000	600 x 400	15,83	15,5	0,33
	1920 x 1200	15,95	16,15	-0,2

Conclusão

- Bom resultado de desempenho

  Técnica não apresentou perda significativa de desempenho.

- Rótulos são montados em tempo real, em GPU.

  Não é preciso possuir texturas de rótulos construídas antes de cada quadro.

  Não é preciso lidar com problemas de troca de contexto de textura.

- Economia de memória.

  Armazenamento apenas de códigos ASCII e atlas.

  E não de uma textura ou várias texturas para rótulos.

- Cálculo de rótulos somente para objetos da tela.

  Independente do número de objetos, cálculo de rótulos só para objetos da tela.

Trabalhos futuros

- Melhorar *antialiasing*
- Posicionamento de rótulos em outras primitivas e malhas.
- Orientação do texto.
- Teste do algoritmo em uma passada.
- Escrever texto com mais de uma linha.
- Explorar técnica fora do domínio de CAD.

Referências

- AVEVAREVIEW 2010. Aveva Home Page. Disponível em: http://www.aveva.com/ Acesso em: abr 2010.
- PRADO, R. D., RAPOSO, A., SOARES, L. P. 2011. Autotag: Applying Tags to Virtual Objects in Real-Time Visualization Systems. 8º Workshop de Realidade Virtual e Aumentada - WRVA, 2011, Uberaba. Anais do WRVA'2011 - 8º Workshop de Realidade Virtual e Aumentada (CD-ROM). Porto Alegre: Editora da SBC, 2011. p. 1-6.
- LEFEBVRE, S., HORNUS, S., AND NEYRET, F. 2005. Texture Sprites: Texture Elements Splatted on Surfaces. In ACM Symposium on Interactive 3D Graphics.
- QIN, Z., MCCOOL, M. D., AND KAPLAN, C. S. 2006. Real-time texture-mapped vector glyphs. I3D '06: Proceedings of the 2006 symposium on Interactive 3D graphics and games, ACM Press, New York, NY, USA, 125–132.
- CIPRIANO, G., GLEICHER, M. 2008. Text scaffolds for effective surface labeling. IEEE Trans. Visualization and Computer Graphics 14(6), 1675–1682 (2008)

FIM