Maurício Bammann Gehling

Normalpaint: Uma ferramenta interativa de pintura de *normal maps* em modelos 3D

Dissertação submetida a avaliação como requisito parcial para a obtenção do grau de Mestre em Computação Aplicada.

Orientadora:
Dra. Soraia Raupp Musse

São Leopoldo

Janeiro, 2007

Dedico este trabalho ao meu pai,
que além de sempre me apoiar em todos os momentos da minha vida,
me ajudou no início do curso, com várias caronas especiais
enquanto me recuperava do acidente.
Obrigado por tudo.

AGRADECIMENTOS

Agradeço a todos os amigos que ajudaram, em tantas madrugadas que trocamos idéias. São muitas pessoas para citar individualmente, mas cada uma de vocês foi essencial para tornar possível este trabalho.

Agradeço aos meus pais, Edemar e Zaíra, por todo o apoio e incentivo.

Agradeço a minha esposa e eterna namorada Zu, pelo carinho e compreensão em tantas ausências no decorrer do curso.

Agradeço a minha orientadora Soraia Raupp Musse, pelas idéias, revisões, ânimo e amizade que foram imprescindíveis para a conclusão desta dissertação.

RESUMO

Este trabalho apresenta um novo modelo de criação e alteração de mapas de normais, usando uma interface intuitiva que se assemelha a um sistema de pintura. Estes mapas são cruciais para aplicações gráficas, processadas em tempo real, que buscam maior realismo visual. Principalmente jogos digitais atuais se utilizam desta tecnologia, permitindo recriar o visual complexo de um modelo geométrico tridimensional com muitos polígonos, em um modelo simplificado. Desta forma o jogo utiliza somente o modelo mais leve para processar, mas consegue renderizar detalhes no objeto que criam a ilusão de complexidade geométrica.

O processo original de criação deste mapa de normal (também conhecido como *normal map*), é muito trabalhoso. Um artista deve criar o modelo otimizado que será utilizado pelo jogo, além de outro modelo que contém tantos detalhes geométricos quantos forem desejados. Este último modelo servirá apenas como base para a criação do *normal map*, utilizando algum *software* 3D que contenha uma ferramenta para este fim. Da comparação entre os modelos, o mapa de normais é criado, sendo então aplicado no modelo mais simples, que é renderizado com o detalhamento que o *normal map* representa.

Este trabalho apresenta um modelo computacional que propõem eliminar conceitualmente a necessidade da criação do modelo geométrico mais detalhado, possibilitando a criação do *normal map* diretamente no objeto. Desta forma, o artista cria somente o modelo mais simples e "pinta" nele o efeito das alterações das normais visualizando o resultado.

Nesta dissertação apresenta-se detalhes do modelo, o protótipo desenvolvido que visa validar o conceito de pintura de *normal maps*, e discute-se os resultados obtidos.

Palavras-chave: *Normal map*, pintura, *engine* 3D, jogos digitais, objetos 3D

ABSTRACT

This work presents a novel model of creation and modification of normal maps, using a intuitive interface similar of a paint system. These maps are crucial to graphical applications, processed in real time, that claims better visual realism. Most digital games use this technology, allowing the recreation of the visual complexity of a dense geometric 3D model, in a simplified model. That way the game utilize only the lighter model to process, but can render the details on the object that create the illusion of geometric complexity.

The creation process of this normal map is very labor intensive. An artist must create the simplified model that will be used in game, besides another model that is as complex as desired. This last model will be used only as a reference to extract the normal map in a 3D software. Then the normal map is applied in the simpler model, that is rendered with the visual detail the normal map represent.

This work presents a computacional model that propose the concept of eliminate the creation necessity of the complex geometric model, allowing the creation of the normal map directly on the object. This way the artist create only the simple model and "paint" on it the effect of the normal alterations visualizing the results.

This dissertation presents the model details, the prototype developed that focus validate the concept of painting normal maps, and discuss the obtained results.

Keywords: *Normal map*, paint, 3D *engine*, digital games, 3D objects

LISTA DE FIGURAS

SUMÁRIO

1 INTRODUÇÃO

O homem sempre procura meios interessantes de entretenimento para preencher seus momentos de lazer. Dentre as várias opções possíveis, o entretenimento digital têm-se mostrado um dos mais populares da atualidade. No ano de 2005 vendeu-se, somente nos Estados Unidos, mais de 228 milhões de unidades de jogos (considerando jogos de computador e videogames), o que representa uma arrecadação de U$7 bilhões. (ENTERTAINMENT-SOFTWARE-ASSOCIATION, 2006. http://www.theesa.com)

A idade média dos jogadores é de 33 anos, o que representa um fato essencial em relação ao tipo dos jogos consumidos pelo mercado. Títulos de ação, luta, esportes, entre outros gêneros, são campeões de vendas. Para agradar este público tão maduro e exigente, a qualidade gráfica apresentada pelos jogos atuais representa um realismo nunca visto até poucos anos atrás. Deve-se este fato ao uso de modernas técnicas de renderização e iluminação dos mundos virtuais, processados por estes jogos. Inclusive a nova geração de videogames, que começou a surgir no final de 2005, inaugura o termo "cinema em tempo real", devido a sua qualidade visual e realismo.

Figura 1: *Brothers in Arms: Hell's Highway*, um dos jogos da nova geração.

Apesar dos jogos ainda necessitarem de modelos poligonais com número restrito de vértices e faces, para possibilitar o processamento em "tempo real", o uso de texturas especiais conhecidas como mapas de normais ou *normal maps* (KILGARD, 2000) contribui de forma expressiva à qualidade final do modelo. Os *normal maps*[1] são texturas que representam a orientação de vetores normais da geometria aplicada a eles. Para cada *pixel* da textura, é feita uma relação da cor à direção de um vetor normal, que é então usado para o cálculo da iluminação do modelo.

Desta forma, um modelo sem muitos polígonos consegue um resultado visual complexo, pois demonstra ter um detalhamento geométrico muito maior, pela ação do *normal map*. Assim têm-se um modelo leve para ser processado pela máquina e visualmente rico para o observador. Este recurso tornou-se padrão no processo de desenvolvimento de jogos que buscam realismo gráfico. A Figura 2 mostra um exemplo de *normal map*.

Figura 2: Exemplo representando uma parede de pedra.

1.1 O problema

Apesar do expressivo resultado visual do *normal map* (conforme será detalhado melhor nos próximos capítulos), o processo de criá-lo é trabalhoso e lento. O artista precisa fazer, além do modelo simplificado que será processado pelo jogo, um segundo modelo altamente detalhado, que será usado como base para gerar o *normal map*. Para esta tarefa, pode ser empregada uma ferramenta, geralmente disponível nos $software$s de modelagem 3D atuais (como Maya ou 3D Studio Max), que gera o *normal map* baseado no modelo detalhado. Basicamente este processo ocorre da seguinte forma: é feita uma projeção do modelo mais simples em relação ao modelo mais detalhado, sendo que os dois devem estar na mesma posição espacial, um dentro do outro. Então os vetores normais do modelo mais complexo são capturados e transformados em cores RGB, de acordo com sua inclinação, formando o *normal map*. Após o fim deste processo, tem-se a textura completa, pronta para ser usada no jogo. Assim, a técnica provê um modelo otimizado para ser processado em tempo real, mas com o detalhamento visual semelhante ao modelo de alta resolução criado como referência. A Figura 3 demonstra que o modelo

[1]Neste trabalho, optou-se por utilizar o termo em inglês pela importância que o mesmo possui na comunidade científica.

com o *normal map* aplicado apresenta qualidade semelhante ao modelo complexo, porém mantendo o número de polígonos do modelo simplificado.

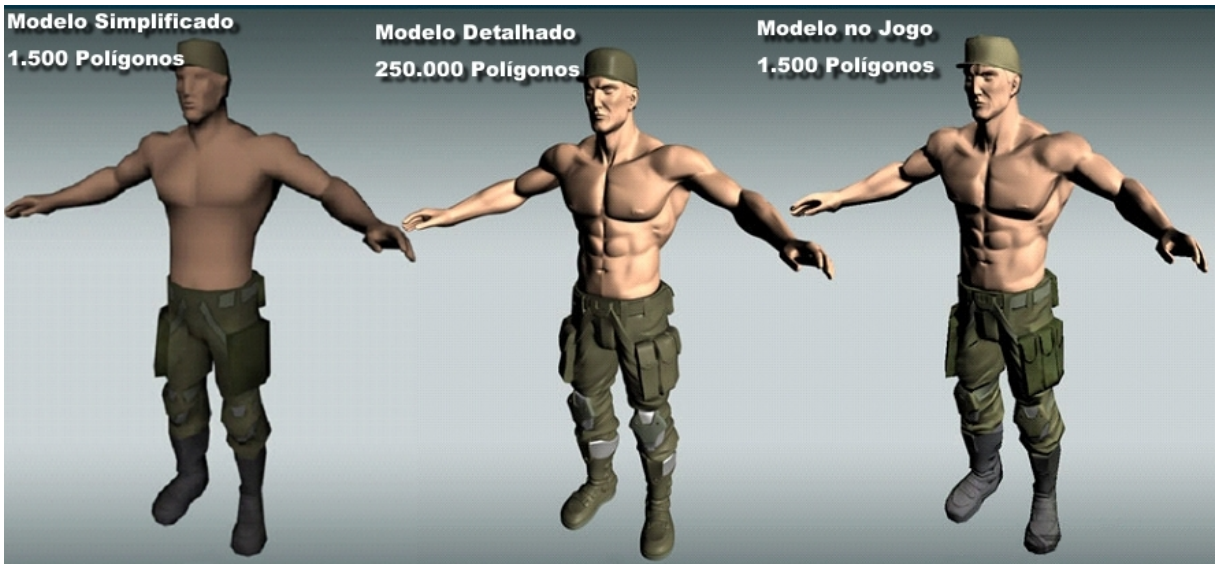

Figura 3: O personagem detalhado é necessário para gerar o *normal map* que será usado no modelo simplificado. (Personagem do jogo *Far Cry* - Crytek Studios)

Todo o trabalho empregado em criar o modelo detalhado, tarefa que pode consumir semanas, serve somente para extrair os detalhes que criam o normal map. Ou seja, há um grande esforço para se chegar neste resultado.

Por mais que a qualidade final do jogo compense esta tarefa, o tempo de uma produção que se baseia amplamente nesta tecnologia pode facilmente triplicar em relação a um projeto que não use este sistema. Desta forma, um dos grandes problemas da técnica é o enorme trabalho manual que ela exige.

1.2 Objetivos gerais

A maior motivação para o desenvolvimento desse trabalho é otimizar e facilitar a criação do *normal map*, procurando diminuir o processo manual que é exigido na sua concepção. Procura-se também prover a possibilidade de manipular o *normal map* de uma forma diferenciada.

Assim o objetivo principal é criar um modelo para criação de *normal maps*, oferecendo o recurso de usar ferramentas que possam criar os relevos desejados sem a necessidade de desenvolver um outro modelo geométrico detalhado, conforme demonstrado pela Figura 3. Este resultado é conseguido utilizando-se um sistema semelhante à "pintura" de normais, onde o usuário configura o tipo de relevo desejado e "pinta" na superfície do modelo simplificado. Então o protótipo simula a inclinação das normais para gerar o efeito e escreve a cor correspondente no *normal map*, tendo o resultado da técnica exibido em tempo computacional interativo.

Assim o artista 3D pode interativamente simular o detalhamento que deseja no

modelo final, sem a necessidade de criar um outro modelo separado para este fim. Ao concluir o procedimento, o protótipo permite salvar em disco o *normal map* criado, que então pode ser usado no jogo.

Aproveitando as possibilidades de experimentação que este modelo possibilita, foi implementado um sistema de animação de *normal maps*, ainda não explorado na literatura.

1.3 Objetivos específicos

Com a finalidade de desenvolver este modelo, definiu-se os seguintes objetivos específicos:

- Propor um modelo que trate a criação e animação de *normal maps* através de ferramentas interativas.
- Desenvolver um protótipo que visualize, em tempo computacional interativo, o modelo geométrico simplificado juntamente com o *normal map* aplicado.
- Propor e desenvolver um sistema de deformações de vetores normais com o objetivo do artista poder criar o tipo de relevo que quiser.
- Desenvolver a animação do *normal map*, em função do tempo.

1.4 Organização do trabalho

O capítulo 2 apresenta a revisão bibliográfica que aborda diversas técnicas de renderização que simulam superfícies irregulares, além de comentar outros trabalhos que utilizam *normal map* para funções específicas. Também ferramentas que permitem pintar texturas em modelos 3D de forma interativa serão apresentadas.

O capítulo 3 apresenta o modelo do trabalho desenvolvido, onde cada aspecto deste será comentado em detalhes. O capítulo 4 apresenta detalhes do protótipo, enquanto o capítulo 5 discute os resultados obtidos. Finalmente, o capítulo 6 traça algumas considerações finais.

2 REVISÃO BIBLIOGRÁFICA

Este capítulo destina-se a apresentar uma visão geral da literatura, relacionada à este trabalho, através de três tópicos definidos a seguir:

- Técnicas de renderização que simulam superfícies irregulares, onde apresenta-se os trabalhos que descrevem a utilização original de *normal maps*;
- Outras utilizações de *normal maps*; e
- Ferramentas que permitem pintar texturas em modelos 3D de forma interativa (podem ser usadas como inspiração neste trabalho).

2.1 Técnicas de renderização que simulam superfícies irregulares

Em computação gráfica, o termo "textura" não é exatamente o que se costuma entender como textura no mundo real. No dia-a-dia, a palavra "textura" se refere a sulcos, ranhuras, saliências e outras irregularidades em superfícies. A noção em computação gráfica de textura é mais semelhante à definição de um adesivo, como se fosse um papel de parede liso que é aplicado no modelo tridimensional. No mundo real, reconhecemos os sulcos e saliências de superfícies por causa da forma como a luz interage com estas irregularidades.

De fato, estas saliências fazem parte da forma geométrica dos objetos na vida real. Em certos casos, a escala destas irregularidades é muito pequena em relação a forma tridimensional geral do objeto. Como por exemplo, esta é a razão porque as pessoas consideram uma bola de *golf* como redonda, apesar de suas concavidades.

Para capturar melhor a noção real de textura e trazê-la para a computação gráfica (aplicando estas ranhuras e saliências que deixam a superfície mais realista) usa-se uma outra textura conhecida como *bump mapping* (BLINN, 1978). Esta técnica criada por Blinn em 1978, consiste em uma abordagem de renderização baseado em textura, para simular os efeitos de iluminação causados pelas irregularidades em superfícies. Pela codificação destes detalhes em mapas de textura, o *bump mapping* simula uma aparência de iluminação irregular no modelo, sem a complexidade e custo de criar os detalhes como perturbações geométricas verdadeiras na superfície. Caso contrário seria muito caro computacionalmente.

O *bump mapping* utiliza uma textura em tons de cinza, com o objetivo de perturbar as normais da superfície como um *"height field"* (campo de altura), funcionando da seguinte forma: para cada *pixel* na textura, o valor de cinza médio (sendo 0 preto e 255 branco) não causa alteração na normal. Cores mais claras simulam uma elevação no relevo, e mais escuras uma depressão. A superfície, sem nenhuma alteração do número de vértices, é então renderizada e iluminada baseada nestas normais perturbadas. Estes cálculos são executados em cada *pixel* visível do modelo.

De forma semelhante em 1984, Cook (COOK, 1984) introduziu uma técnica chamada *displacement mapping*. Esta técnica usa uma textura para determinar o quanto grande é a distância geométrica ao longo do vetor normal do pixel. Neste caso a superfície é de fato subdividida, assim observando-a em um ângulo de visão quase perpendicular à normal da mesma, será exibido sua silhueta correta, por causa da subdivisão. Isto não ocorre com o *bump mapping*, pois a ilusão de detalhamento visual só de dá pela alteração das normais, sem modificar a superfície.

Porém, o processamento necessário para o *bump mapping* como originalmente criado por Blinn, é bem mais custoso que o uso de textura tradicional de cor em um modelo. Várias tentativas foram feitas para reformular-se o *bump mapping* em uma forma eficiente para implementação via hardware. Em geral estas tentativas tiveram várias limitações. Uma alternativa eficiente é adicionar as normais na textura, codificadas de acordo com as cores dos *pixels* da imagem, neste caso chama-se *normal map*. Este conceito foi originalmente apresentado por Fournier em 1992 (FOURNIER, 1992), sendo após adaptado para rodar em hardware por Kilgard em 2000 (KILGARD, 2000). Assim tem-se como reproduzir diretamente a inclinação da normal, sem a necessidade do processamento custoso do *bump mapping* original.

Para o *normal map*, a codificação da direção do vetor normal ($\vec{n}$) é feita atribuindo valores (de -1 até 1) para os componentes nx, ny e nz. Estes valores equivalem as componentes rp, gp, bp, (de 0 até 255) sendo

$$\vec{p} = \begin{pmatrix} r \\ g \\ b \end{pmatrix};$$

a cor de cada *pixel*, desta forma:

$$rp = \frac{(nx+1)255}{2},$$

$$gp = \frac{(ny+1)255}{2},$$

$$bp = \frac{(nz+1)255}{2}.$$

Assim, o *normal map* informa precisamente a direção correta de cada vetor normal, para cada *pixel*, gerando um resultado expressivo na renderização final. O cálculo da luz, nesta superfície que teve suas normais perturbadas pelo *normal map*, resulta numa boa qualidade de iluminação pois tem-se toda a informação do ângulo correto de cada vetor normal.

Outro método de renderização é o *relief texture mapping* (OLIVEIRA; BISHOP; MCALLISTER, 2000), que consegue simular efeitos de *"parallax"* atingindo um ótimo realismo visual. O termo *"parallax"* é atribuído ao fenômeno de que uma superfície que está mais distante que outra, aparenta se mover com velocidade diferente quando vista por outro ângulo. Ou seja, em uma parede de tijolos, a parte interna formada pelo cimento aparenta se mover mais lentamente que os tijolos em si, quando o espectador observa a cena mudando sua perspectiva de visão. Para atingir este efeito, o *relief mapping* utiliza mais um canal na textura. Além do *normal map* codificado nos canais RGB da imagem, há no canal alpha (canal extra da imagem, geralmente usado para criar transparência) uma imagem em tons de cinza que representa um *"height-field"* (mapa de alturas). Este é o responsável pelo efeito parallax empregado na técnica, pois assim são informados quais partes do modelo estão à frente das outras.

Esta técnica apresenta algumas vantagens visuais em relação ao *normal map* tradicional, como a possibilidade de gerar sombras nos relevos e até calcular corretamente a silhueta do modelo. Com o *relief mapping* os jogos podem apresentar mais realismo, pois a simulação de geometria por texturas atinge um nível de qualidade ainda melhor, enriquecendo o mundo virtual do jogo.

Porém este processo necessita de mais poder computacional, e o uso em tempo real de *relief mapping* (POLICARPO; OLIVEIRA; COMBA, 2005) ainda não foi adotado como padrão pela indústria de jogos. Implementações atuais conseguem atingir boa performance, desde que rodem em placas gráficas de última geração, o que implica em um público consumidor muito menor. Por isso, mesmo com a ótima qualidade do *relief mapping*, o *normal map* ainda é o mais usado no cenário atual de desenvolvimento de jogos.

2.2 Outros trabalhos que utilizam *normal maps*

Os *normal maps* podem ser utilizados para os mais diversos fins. Além da sua importância para o desenvolvimento de jogos modernos que procuram maior detalhamento visual sem perder performance, outros aplicativos com objetivos diversos podem contar com esta textura especial para simular geometria.

Filmes de animação 2D tradicionais podem se beneficiar desta tecnologia também, auxiliando na área de colorização. O *normal map* pode ser usado para estimar o volume de um desenho 2D, possibilitando a renderização deste desenho simulando efeitos de luz e sombra (BEZERRA, 2005). O artista faz o desenho de cada quadro da animação a mão, que são escaneados e processados. Então, de acordo com a direção das linhas do desenho, são estimadas as direções das normais. Isso é feito através de algoritmos de análise da estrutura morfológica das curvas, sendo que o vetor normal de cada ponto é determinado tomando-se o vetor perpendicular ao vetor direção da curva. A direção da normal que aponta para fora do interior da curva é escolhida, possibilitando posteriormente um

processo de suavização e interpolação de maneira a gerar um campo de vetores normais convexo. Assim, o desenho terá uma aparência correta de volume. O *normal map* final é composto da interpolação de todas as regiões não sobrepostas que formam a imagem.

Então de posse deste *normal map*, o desenho é renderizado criando a sensação de volume que o *normal map* informa, podendo escolher diversos tipos de renderizações diferentes, foto-realistas ou estilizadas, gerando os brilhos e sombras respectivos de cada um. Desta forma o lento processo de colorização do desenho feito a mão é agilizado.

Figura 4: Processo de colorização automática de desenho. Primeira imagem é o traço original do artista, após dois estilos diferentes de sombreamento automático. (BEZERRA, 2005)

Ainda sobre trabalhos referentes a desenhos animados, o *normal map* foi aplicado em um protótipo que desenha fumaça com visual estilizado em tempo real (MCGUIRE; FEIN, 2006). Um sistema de partículas que usa somente três polígonos por partícula, gera a fumaça com silhuetas e sombra próprias. Para gerar as texturas dos segmentos da fumaça, há um pré-processamento inicial. Primeiramente algumas esferas geométricas aleatórias criam a forma básica de um pequeno segmento da fumaça. Este processo é repetido quatro vezes para haver variação. A partir delas são gerados *normal maps* e mapas de profundidade, compondo uma textura RGBA (canais de cor RGB mais canal Alpha). Ainda, é criada mais uma textura RGBA que contém a cor desejada do segmento da fumaça nos canais RGB, e o recorte do mesmo no canal Alpha.

As esferas geométricas são então descartadas e somente as texturas são usadas com as partículas. Desta forma, lendo as informações destas duas texturas, a renderização das partículas consegue recriar uma simulação de fumaça ou nuvens com volume e sombras, no estilo de desenho animado. Observe na Figura 5, a imagem resultante do processo. Este trabalho pode ser útil em aplicações como jogos, filmes ou produções de história em quadrinhos, podendo compor o desenho manual do artista com elementos desenhados pelo computador.

O *normal map* ainda pode ser empregado para aumentar o realismo da renderização da pele em personagens virtuais, com o objetivo de reaver os detalhes sutis dos poros e demais rugosidades que a pele possui. Estes detalhes só são visíveis quando o personagem é visto de perto, mas contribuem efetivamente para atingir um maior realismo. Para criar um *normal map* fisicamente correto, foram gerados alguns pequenos moldes em silicone da pele humana (HARO; ESSA, 2001). Os moldes foram extraídos de diferentes partes do rosto, como nariz e testa, afim de obter as variações características da pele facial. Cada molde

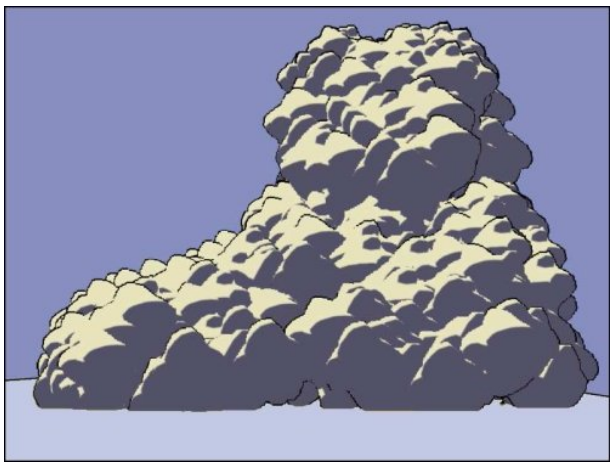

Figura 5: Fumaça com estilo visual de desenho. (MCGUIRE; FEIN, 2006)

não era maior que o tamanho de uma moeda. Foram gerados *normal maps* de acordo com o sombreamento da superfície destes moldes (HORN; BROOKS, 1989). Então estas texturas são combinadas e propagadas para formar uma textura maior, que será usada para renderizar o modelo do rosto de um personagem virtual. Este modelo consegue ser renderizado em tempo real. A Figura 6 demonstra o resultado atingido.

Trabalhos nas áreas de manipulação e otimização de malhas tridimensionais também podem se beneficiar do uso de *normal maps*. Por exemplo, equipamentos leitores de objetos físicos (scanners 3D) produzem modelos computadorizados de alta resolução, mas geralmente apresentam algum ruído na superfície da malha. Essas perturbações podem atrapalhar a análise correta da superfície, gerando uma visualização mais irregular do que o próprio objeto original demonstra. Para resolver este problema, é possível filtrar as normais da superfície, ao invés de filtrar a posição dos pontos da malha (TASDIZEN et al., 2003). Esta ferramenta de processamento geométrico ocorre em duas etapas. Primeiro opera no *normal map* da superfície e após manipula a superfície para se adequar as normais processadas. Esta estratégia permite uma grande gama de operações de processamento topológico em modelos, incluindo suavização que preserva arestas, entre outros tipos de filtragem. Além disso a forma genérica desta implementação a faz apropriada para modelos com superfícies altamente complexas, como estas construídas diretamente por dados de medição.

2.3 *Softwares* que permitem pintura em modelos tridimensionais

Esta seção não trata de *normal maps*, mas foi incluída neste trabalho por poder servir como inspiração para o protótipo desenvolvido.

A aplicação de textura bidimensional em modelos tridimensionais é uma prática comum em aplicações em tempo real, para aumentar o realismo do modelo. Para desenvolver esta textura o artista deve usar um *software* de edição de imagens, para somente depois aplicar o resultado em um modelo 3D, podendo observar como a imagem se comporta

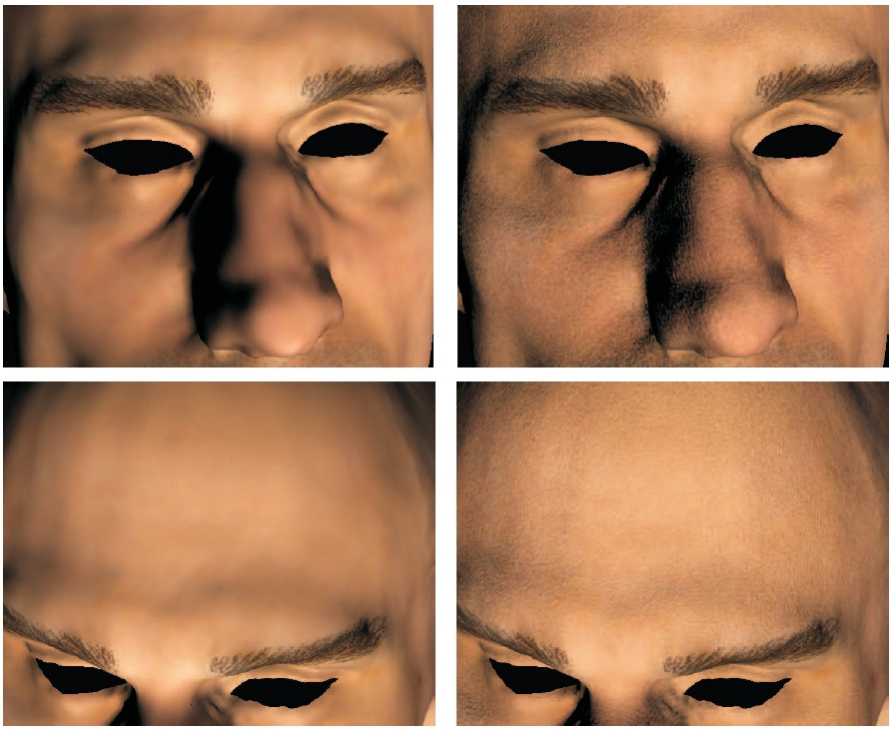

Figura 6: As imagens na esquerda não possuem os detalhes recriados com o *normal map*. Na direita nota-se o efeito.(HARO; ESSA, 2001)

projetada no objeto, de acordo com suas coordenadas de mapeamento. Esse processo é lento, pois mesmo para se fazer um simples teste do resultado final, deve-se salvar a textura em disco, atualizar a imagem no *software* 3D, e então finalmente ver o efeito no modelo.

Para amenizar este problema, novas aplicações que permitem pintar diretamente sobre modelos 3D foram apresentadas. Um dos primeiros trabalhos nesta área é o de Daily e Kiss (DAILY; KISS, 1995). Também surgiram técnicas que buscam aumentar a interação entre o artista e o modelo, simulando ferramentas com periféricos intuitivos (*force-feedback*), como em (AGRAWALA; BEERS; LEVOY, 1995) e (GREGORY; EHMANN; LIN, 2000).

Este tema atraiu a atenção da indústria de *softwares* de computação gráfica, sendo que vários sistemas atuais contém ferramentas para pintura direta em modelos tridimensionais. Estas soluções estão incorporadas nos principais *softwares* de modelagem 3D, e existem até outros programas desenvolvidos especificamente para este propósito.

Um dos principais *softwares* de modelagem e animação do mercado é o Maya (desenvolvido pela Alias, recentemente adquirida pela Autodesk[1]), que possui recursos avançados como simulação e renderização de pêlos, tecidos, líquidos, cabelos longos, partículas, entre outras funções sofisticadas. Sua interface é reconhecida como uma das mais organizadas e eficientes entre diversos *softwares* de modelagem. O Maya possui um recurso chamado Artisan, que são ferramentas que usam uma interface intuitiva ba-

[1]http://www.autodesk.com

seada num sistema de pintura e escultura para realizar uma grande gama de tarefas de modelagem, animação, renderização e efeitos.

Há várias ferramentas Artisan diferentes, sendo que todas funcionam como pincéis de pintura. Os pincéis permitem esculpir modelos (empurrando ou puxando vértices), pintar seleções de componentes de uma malha para aplicar outras modificações, pintar atributos (de deformação, de fluidos, pêlos, etc), além da pintura direta em modelos 3D. Estas ferramentas ainda são sensíveis a pressão, se o usuário, por exemplo, usar uma tablet (hardware que permite interagir com o programa usando uma pequena superfície sensível ao toque de uma caneta especial).

No caso da pintura em 3D do Maya, devemos primeiramente escolher uma textura para aplicar ao modelo, após selecionar uma cor e começar a pintura. Esta cor será misturada com a textura existente, de acordo com a escolha do método de mistura que o usuário deseja (as opções incluem multiplicar os *pixels*, usar a cor mais clara ou escura do resultado da pintura com a textura de fundo, entre outras). O Maya permite ajustar o tamanho do pincel que será usado, sua forma e opacidade. Assim o usuário tem amplo controle sobre como será feita a pintura. A Figura 7 mostra uma tela do Maya.

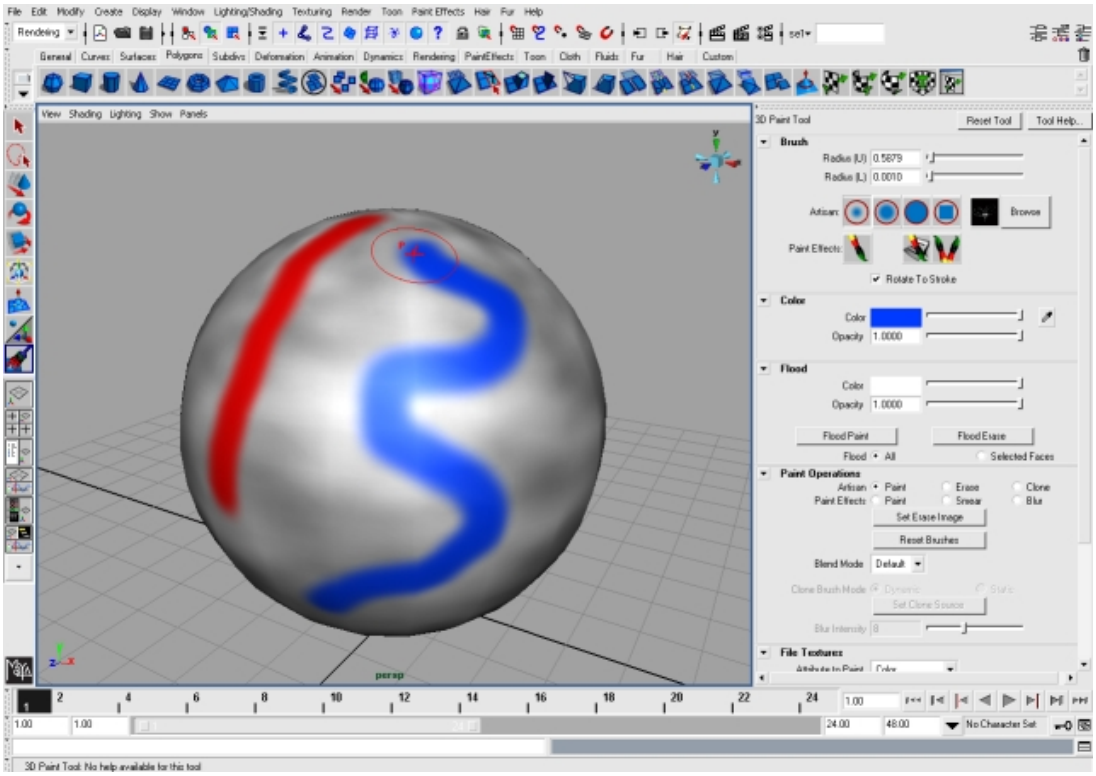

Figura 7: Interface do Maya com a ferramenta de pintura 3D acionada.

Dentre os *softwares* que existem especialmente para realizar a pintura direta em modelos 3D, um dos mais conhecidos é o Deep Paint 3D, desenvolvido pela empresa Right Hemisphere[2]. Ele não possui ferramentas de modelagem ou animação, só permite importar modelos criados em outros *softwares*. Sua funcionalidade é totalmente focada na edição do material deste modelo importado, carregamento de mapas de textura e pintura feita

[2]http://www.righthemisphere.com/

diretamente no modelo 3D. A utilização é semelhante ao módulo de pintura do Maya, como descrito anteriormente. O Deep Paint 3D ainda conta com uma grande quantidade de tipos de pincéis selecionáveis, alguns imitando tintas reais, como óleo, acrílico, etc.

Este tipo de aplicativo agiliza o fluxo de trabalho permitindo uma maior integração com os principais *softwares* de modelagem, através da possibilidade de usar plug-ins que atualizam o modelo 3D em ambos os programas. Também realiza integração semelhante atualizando texturas com o Photoshop, que é o principal editor de imagens 2D do mercado. Porém mesmo com todos estas funcionalidades, seu uso se restringe a pintura (diretamente em modelos 3D ou ainda usando a opção de pintar numa tela 2D), sendo que toda modelagem e renderização final deve ser feita externamente em outro aplicativo. A Figura 8 mostra uma tela do Deep Paint 3D.

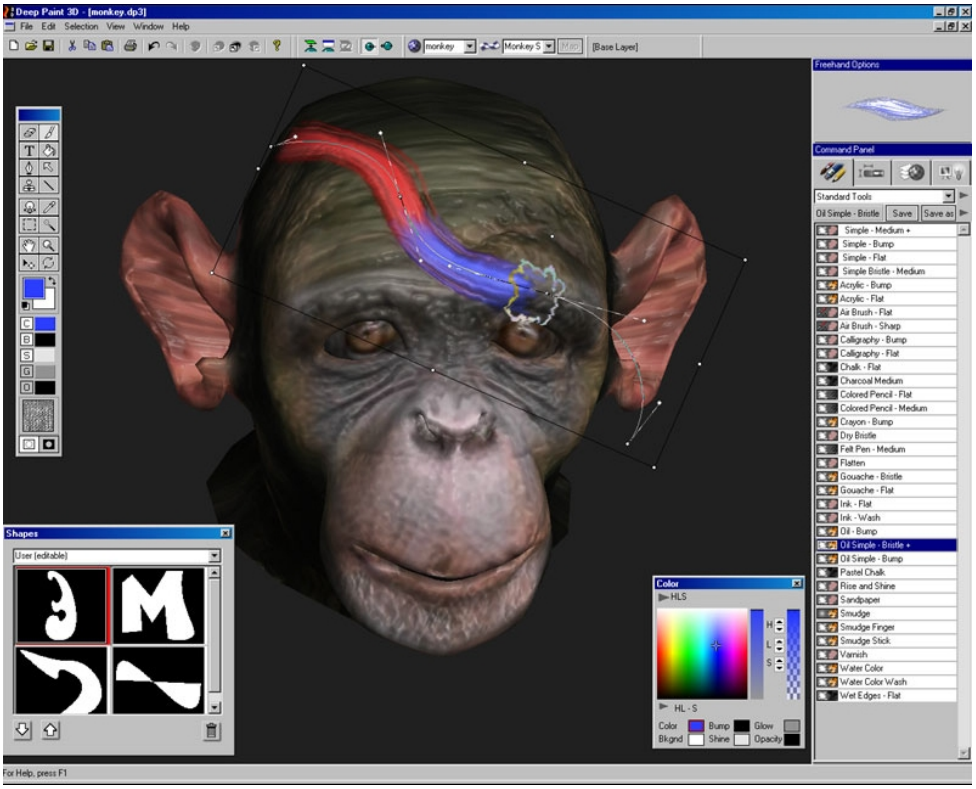

Figura 8: Interface do Deep Paint 3D. (Imagem retirada do site http://www.righthemisphere.com)

Outro *software* que utiliza o conceito de esculpir a malha, utilizando ferramentas semelhantes a pincéis é o Zbrush, desenvolvido pela Pixologic[3]. Ele é mais utilizado para esculpir subdividindo a geometria, podendo criar não somente detalhes maiores como também sutis. Por exemplo rugas no rosto de um personagem, rugosidades de sua roupa ou relevos da musculatura, entre outros detalhes, são todos possíveis de serem desenvolvidos com o Zbrush. Para isso ele cria e administra modelos de altíssima densidade, podendo chegar a milhões de polígonos. Possui uma tecnologia de otimização que procura atingir interação em tempo real com estes modelos.

Também oferece um recurso chamado Zspheres, que auxilia a geração de uma

[3]http://www.pixologic.com

malha inicial em baixa resolução. Com esta ferramenta o artista posiciona esferas que se conectam entre si na cena 3D, formando a malha. Esta malha então serve como base para ser subdividida e detalhada através de suas extensas ferramentas de escultura. Além disso a pintura de cor sobre o modelo, como supracitada nos outros *softwares*, também está presente no Zbrush.

Depois da modelagem pronta, o usuário ainda pode gerar uma textura de *displacement* ou um *normal map*, que aplicada ao modelo de baixa resolução, poderá reconstruir o detalhe da malha mais densa na renderização final. Por estas facilidades de modelagem e velocidade de interação, o Zbrush está sendo cada mais utilizado na realização de filmes, especialmente para gerar efeitos especiais, entre outras produções que utilizam com freqüência personagem virtuais detalhados. A Figura 9 mostra uma tela do Zbrush.

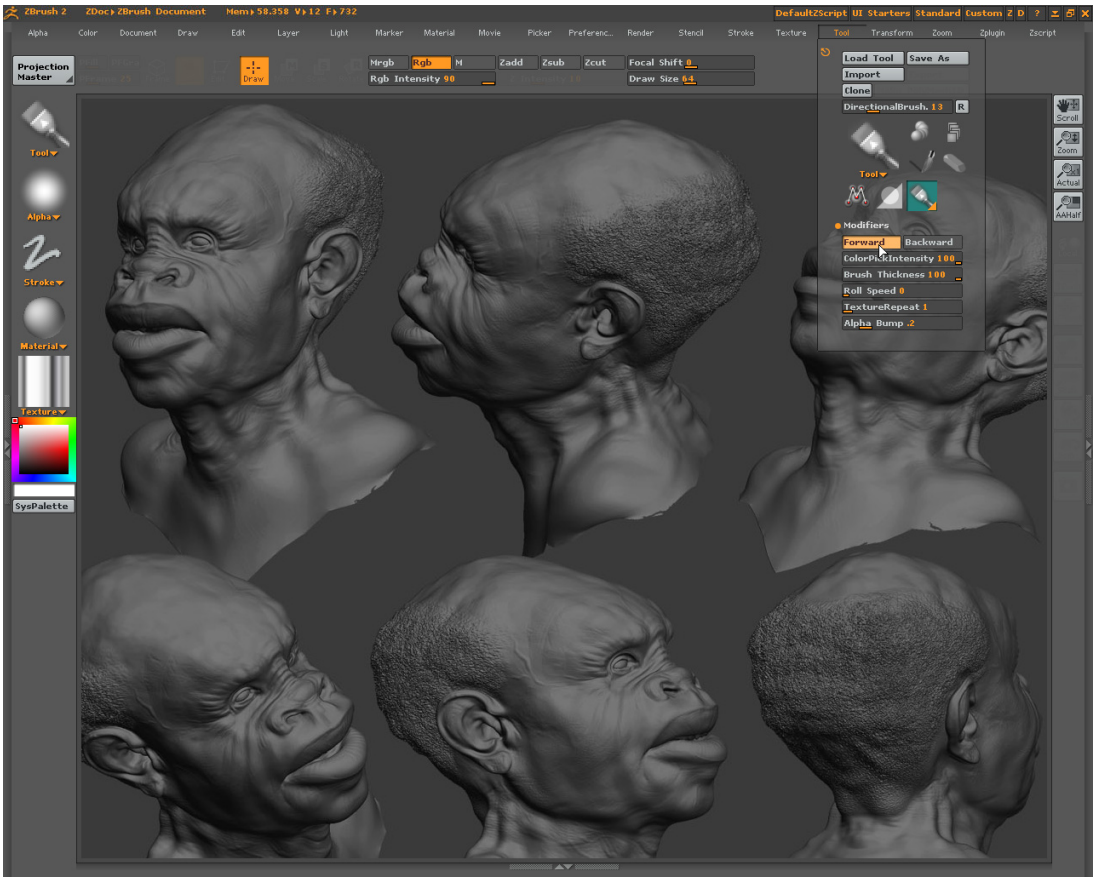

Figura 9: Tela do Zbrush. (Imagem retirada do site http://www.pixologic.com)

A vantagem da pintura de uma textura ser feita diretamente no modelo 3D é evidente. O usuário assiste na hora o resultado de seu trabalho aplicado ao modelo. Porém apesar da pintura direta auxiliar o trabalho do artista, o uso de um editor tradicional de imagem, na maioria dos casos, ainda se faz necessário. O mais tradicional deles é o Photoshop, desenvolvido pela Adobe[4], que conta com recursos sofisticados de pintura, manipulação da imagem com alterações de cor, recortes, composição entre várias camadas, máscaras de seleção, entre outras ferramentas.

[4]http://www.adobe.com

É complexa a produção de texturas de alta qualidade que contenham detalhes pequenos diretamente em um objeto 3D, pois a curvatura e sombreado do modelo atrapalham a análise e alteração da textura. Além disso, o uso de várias camadas de imagens para desenvolver a textura é comum no trabalho de um artista, pois auxilia muito na obtenção de um melhor resultado através da composição, permitindo a alteração futura dos elementos que formam a textura, que estão de forma organizada separados em camadas distintas. A manipulação e controle de várias camadas necessita uma interface mais dedicada a isto, outro fator que complica a integração da edição 2D com o modelo 3D. Desta forma o uso de pintura direta em modelos 3D fica mais restrito a retoques da textura, e não no desenvolvimento completo da mesma.

Assim, os diversos *softwares* que oferecem este recurso de pintura direta, trazem alguma facilidade ao processo de criar e visualizar texturas aplicadas a um modelo 3D. Porém na maioria dos casos não substituem o uso de um editor especializado em manipular imagens 2D. Além disto a possibilidade de criar *normal maps* diretamente com ferramentas deste tipo, sem precisar esculpir um modelo de alta densidade, não está atualmente disponível em nenhum *software* que permita a pintura direta em objetos 3D.

2.4 Contexto deste trabalho no estado-da-arte

Mesmo com todas as facilidades alcançadas pelos *softwares* citados neste capítulo, a modelagem de um objeto 3D em alta resolução é necessária para servir como base à criação do *normal map*. Lidar com este modelo, que pode atingir milhões de polígonos, pode consumir uma quantidade expressiva de recursos de máquina, como memória e hardware acelerador gráfico. Além disso, para o uso em tempo real do modelo simplificado com o *normal map* aplicado, esta modelagem de referência deve ser feita com cuidados. Em certos casos, detalhes muito sutis ou ângulos retos entre as faces podem não ser notados no resultado final da renderização usando o *normal map*.

Desta forma, seria mais eficiente poder visualizar o modelo já com o *normal map* aplicado, conferindo a qualidade final da renderização. Um protótipo que permita a alteração de normais, através de uma interface intuitiva, direto no objeto 3D, é o objetivo final deste trabalho. Assim o artista possui a facilidade de alterar as normais de forma interativa, observando o resultado do efeito no modelo simplificado, que é exatamente o mesmo que será usado na aplicação destino. Além disso não será necessário o processamento de outro modelo de referência com grande densidade de polígonos.

Outras investigações realizadas neste trabalho exploram animações de *normal maps* (alterações em função do tempo). Ainda nenhum aplicativo usou, de qualquer forma, esta animação de *normal maps*, nem mesmo a possibilidade de serem manipulados diretamente como descrito anteriormente.

3 MODELO DE PINTURA DE *NORMAL MAPS*

Neste capítulo é apresentado o modelo desenvolvido neste trabalho. Cada parte que compõe o modelo será apresentada nas seções seguintes. A Figura 10 exibe uma visão geral do mesmo.

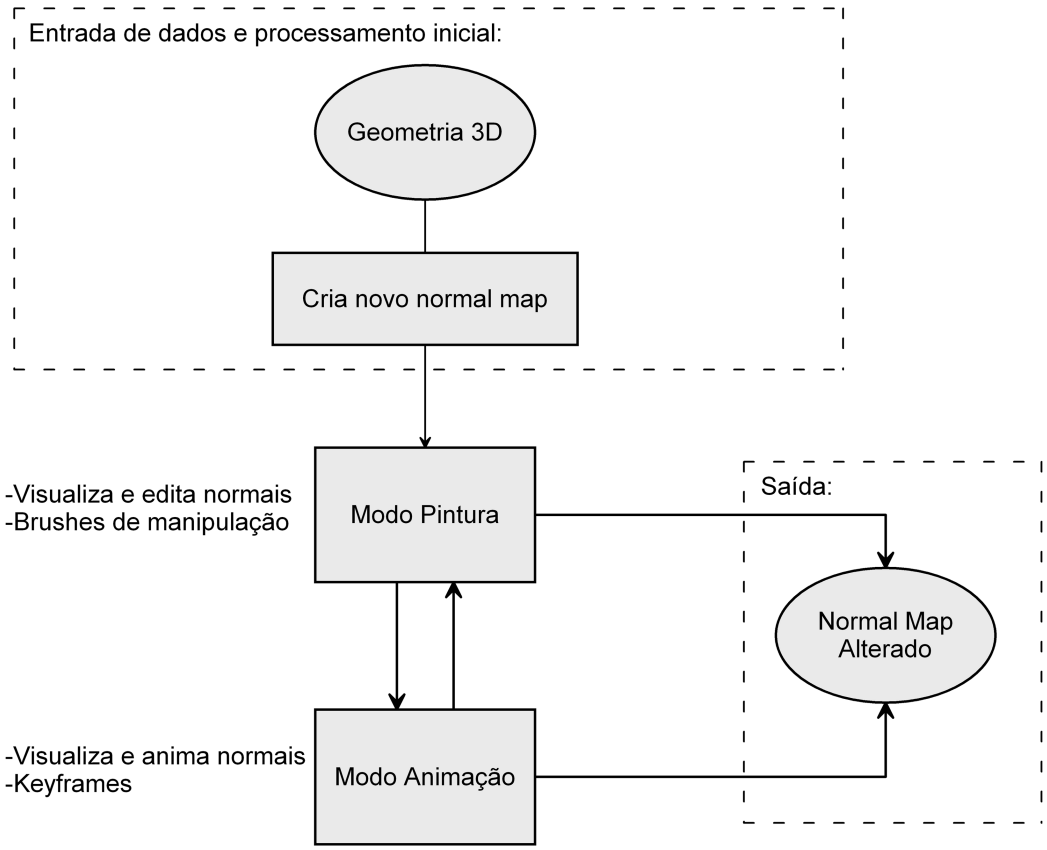

Figura 10: Modelo desenvolvido.

Conforme visto anteriormente, o uso tradicional do *normal map* no desenvolvimento de um jogo, é destinado a criar ilusão visual de complexidade geométrica em objetos 3D simplificados. Para isto um segundo objeto 3D que servirá somente como base para a criação automática do *normal map*, deve ser criado pelo artista. Desta forma ele

esculpe os detalhes desejados que serão codificados no *normal map* pelo *software* 3D escolhido.

A idéia central deste trabalho está focada na possibilidade de gerar o *normal map* por um método mais direto, sem a necessidade de elaborar um outro objeto 3D como referência. A primeira pergunta que tentou-se responder foi: "O que significa gerar *normal maps*?" Sendo a decisão gerar *normal maps* através de sistemas de pintura, ou seja a geração dos *normal maps* neste trabalho se dá através de pinceladas, utilizando mouse ou outro dispositivo de interação. Assim uma interface baseada em sistema de pintura foi utilizada. O objetivo é permitir a simulação da geometria que será codificada pelo *normal map*, "pintando" diretamente sobre o modelo simplificado que será utilizado e processado em tempo-real pelo jogo.

A seção 3.1 descreve a entrada de dados do modelo e o processamento inicial. As partes centrais do modelo constituem-se na manipulação do *normal map*, através da "pintura" (descrita na seção 3.2) e animação usando *keyframes* (descrita na seção 3.3). Por fim na seção 3.4 será comentada a saída do protótipo, ou seja, o *normal map* final resultante das alterações realizadas (ou vários *normal maps* no caso de ser feito uma animação).

3.1 Entrada de dados - Geometria 3D

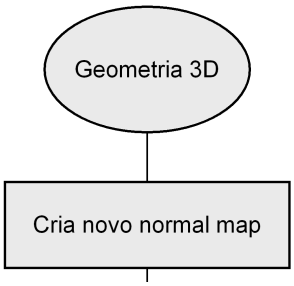

Figura 11: Parte do modelo referente a entrada de dados e processamento inicial.

O fluxo de produção da parte artística de um jogo digital pode ser composto pela modelagem do objeto 3D, edição do *normal map* e exportação para o jogo em si. Desta forma o artista primeiramente usa um *software* comercial de modelagem, otimizando o modelo para ter poucos vértices e assim permitindo que seja processado em tempo real pelo jogo. Então este objeto 3D é carregado no protótipo, possibilitando a sua renderização e criação de *normal maps*, em tempo computacional interativo.

Para a modelagem 3D, vários *softwares* comerciais estão disponíveis, como o Maya citado no capítulo anterior. Além dele existem muitos outros como por exemplo 3D Studio MAX da Autodesk[1], Softimage da Avid Technology[2] e LightWave 3D da NewTek[3]. Ainda há opções gratuitas como o Blender[4], que é um *software* livre, com código aberto.

[1] http://www.autodesk.com
[2] http://www.avid.com
[3] http://www.newtek.com
[4] http://www.blender.org

Um *normal map* inicial é automaticamente criado, servindo como base para as futuras alterações. Este *normal map* é somente um mapa de textura contendo uma só cor azulada, próximo ao lilás, pois representa todos os vetores normais apontando para frente, ou seja, apontando para o observador. Esta cor é formada pelo RGB=(128,128,255), pois a cor demonstra o eixo que a normal está apontando. Para uma cor qualquer (R,G ou B) com valor 0, tem-se a coordenada de orientação normalizada do vetor como -1; para cor 128, tem-se a coordenada $= 0$; e por fim para a cor 255, tem-se a coordenada $= 1$. Assim, como o *normal map* inicial não representa nenhuma deformação na orientação padrão dos vetores normais, ou seja, todos estão perpendiculares em relação as faces da geometria, temos a coordenada representada por [0,0,1]. Isso significa que reside no plano X e Y, mas aponta para frente no plano Z. Transformando em cor RGB temos (128,128,255), como citado anteriormente. Segue a definição de *normal map* inicial.

Definição 1 *Um normal map inicial* $< nmi >$ *é descrito através de uma matriz de cores* R, G, B *que identificam a orientação do vetor normal, em* x, y, z.

3.2 Modo pintura

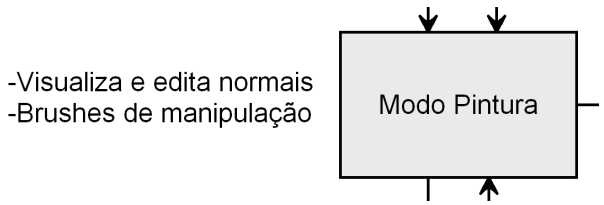

Figura 12: Parte do modelo referente manipulação das normais.

Esta é uma das partes principais do modelo. Aqui é feita a manipulação das normais, atualizando o *normal map* existente. Para isso são usadas ferramentas que irão "pintar" relevos representados pelo *normal map*, sendo visíveis na renderização do objeto 3D.

O sistema de pintura, proposto neste trabalho, funciona na superfície do modelo simples, ou seja, é possível pintar em diferentes partes de uma face sem a necessidade de haver mais vértices subdividindo a face. O pincel deve ser lido pelo sistema de forma dissociada à quantidade de vértices existente no objeto 3D. Para isto ocorrer, é necessário a aplicação de um *grid* virtual sobre as faces, fazendo uma subdivisão lógica e permitindo que seja registrada a posição relativa do pincel dentro da face. Então, independente do número de faces ou vértices, a forma pincelada pelo usuário será interpretada corretamente.

Quando um artista cria geometria para desenvolver o modelo de referência, no processo tradicional de criação do normal map, uma das ferramentas mais utilizadas é o *"extrude face"*. Este método cria um novo conjunto de faces, em posição relativa a face anterior que originou o processo. Estas novas faces podem ser movimentadas ou escaladas, criando um relevo na malha (Figura 13). Baseado nisto, no modelo desenvolvido neste trabalho, a cada pincelada do usuário é simulado o resultado final equivalente a um *"extrude face"*, delimitado pela forma demarcada com o pincel.

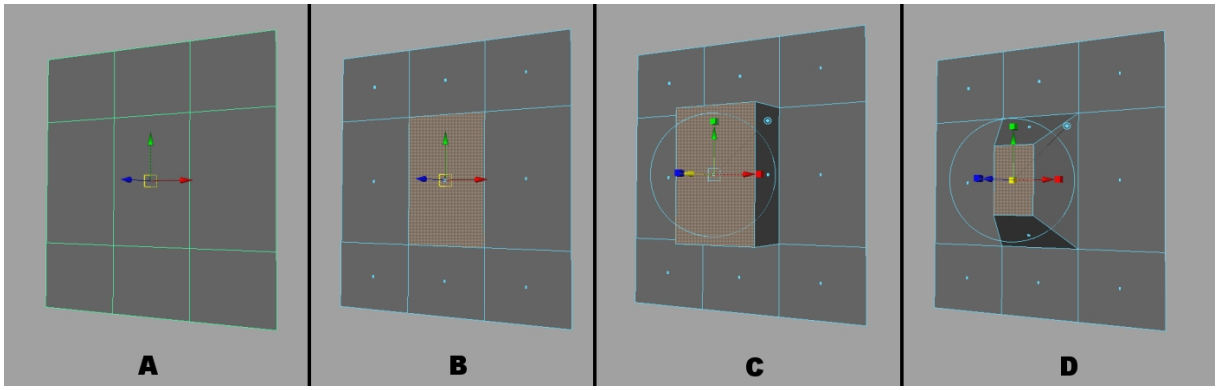

Figura 13: A) Plano poligonal com algumas subdivisões. B) Face selecionada. C) *"Extrude face"* executado na face selecionada. D) Nova face escalada.

O *"extrude face"* é caracterizado no *normal map* pela alteração dos vetores normais na borda da forma realizada. Assim, a parte central da forma pincelada não sofre alteração de cor em relação ao *normal map* padrão (que são todas as normais apontando em direção ao observador), pois ela não representa nenhuma inclinação. Porém, a área relativa a extremidade da forma, representa a inclinação que delimita "rampas" ligando a parte interna à área externa da pincelada. Assim, a área mais colorida do *normal map* será somente na borda da forma pintada, pois é o único local onde houve inclinação das normais (Figura 14).

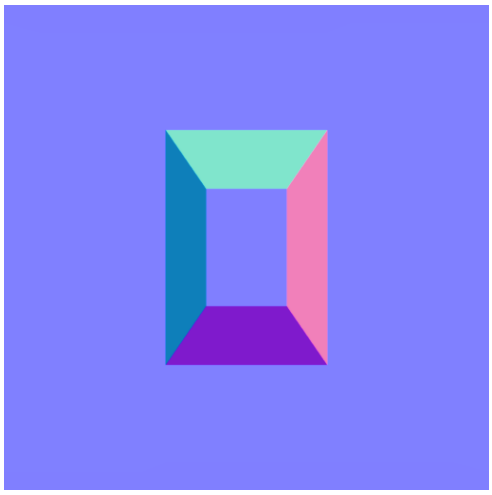

Figura 14: *Normal map* referente a malha do exemplo anterior (Figura 13 - D).

Além disso, quando o artista modela subdividindo o objeto 3D com a ferramenta *"extrude face"*, pode redimensionar as novas faces criadas da forma que quiser, podendo fazer uma alteração sutil ou drástica no tamanho da face. Isto reflete no *normal map* da seguinte forma: a área colorida, que representa a inclinação das normais vizinhas à face central, ficará maior ou menor de acordo com a escala aplicada (Figura 15).

Assim, o modelo desenvolvido neste trabalho visa recriar o efeito gerado através do *"extrude face"* no *normal map*, inclinando as normais na borda da área pintada. O

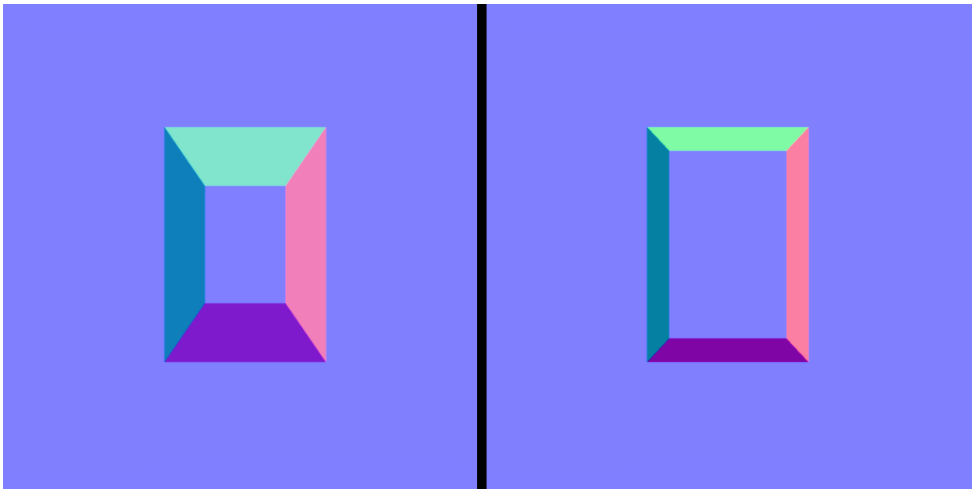

Figura 15: *Normal maps* representando a troca de escala da face central da malha de referência.

usuário pode configurar que tamanho terá esta área simulada do *"extrude face"*, recriando a ação do artista que escala as novas faces de acordo com sua vontade. O objetivo é criar um *normal map* que seja semelhante ao processo tradicional usado pelo artista, que usa ferramentas de subdivisão para esculpir o modelo de referência, criando novas faces.

3.2.1 Definições do sistema de pintura

O modelo desenvolvido neste trabalho sugere que a geração de normal maps seja feita através de sistemas de pintura. Seguem as definições:

Definição 2 *Pincelada* $< p >$ *é definida por sua forma central* $< fc >$ *e borda* $< b >$*. A pincelada ocorre sobre o objeto 3D e cria relevos no normal map de acordo com sua forma.*

Definição 3 *A partir da pincelada* $< p >$*, é detectada automaticamente a borda* $< b >$ *da forma pintada, que por sua vez delimita a extremidade da forma central* $< fc >$*.*

É desejável que as pinceladas possam sofrer variações com o intuito de poder modelar diferentes efeitos. Assim, as pinceladas podem variar de acordo com seus atributos, a seguir definidos:

- Área do pincel $< ap >$: Configura o tamanho da área central afetada pela ferramenta, é equivalente ao tamanho ou grossura do "pincel".
- Direção do relevo $< dr >$: Pode ser escolhida a direção que será formado o relevo. A primeira opção seria uma deformação que cria relevo convexo, ou seja, cria volume na direção frontal do objeto 3D. A segunda opção seria criar depressões, ou seja um relevo côncavo. Assim o usuário pode criar formas que simulem o processo de esculpir um objeto em direção ao seu interior ou exterior.

- Área de *extrude* simulado $< ae >$: Configura a área máxima de efeito da ferramenta, até onde ela irá afetar as normais vizinhas a partir da área central que está sendo pintada $< fc >$, simulando assim o efeito relativo a diferentes escalas de faces resultantes do *extrude face*, conforme Figura 15.

As próximas seções apresentam detalhes do modelo no que tange suas variáveis, definidas anteriormente.

3.2.1.1 Detecção da borda $< b >$

Para gerar *normal map* a partir da forma pincelada pelo usuário, é necessário analisar a sua topologia. Como o modelo é baseado em forma central $< fc >$ e bordas $< b >$ da pincelada, precisa-se primeiramente detectar estas informações da pincelada.

Inicialmente, devem ser detectados quais os *pixels* que pertencem a borda $< b >$ da pincelada $< p >$. Esta informação é crucial para a determinação das cores e locais onde elas devem ser pintadas dentro do *normal map*. O algoritmo de detecção utilizado (JAIN RANGACHAR KASTURI, 1995) inicia a partir da detecção do primeiro ponto da borda, conforme segue:

```
begin
    //Percorre toda matriz
    for i ← matriz_largura; i > 0; i − − do
        for t ← 0; t < matriz_altura; t + + do
            if matriz[i][t] == 1 then
                //No índice 0 do array será guardado o primeiro ponto
                borda_x[0] ← i
                borda_y[0] ← t
            end
        end
    end
end
```

Algorithm 1: Detecção primeiro ponto da borda

Faz-se uma varredura para descobrir o primeiro ponto da borda. A procura acontece percorrendo-se as posições da matriz que guardam informações sobre a forma pintada pelo usuário. Cada posição desta matriz binária representa um *pixel* da imagem que guarda a forma da pincelada. (Figura 16)

A varredura segue do primeiro *pixel* da imagem (superior esquerdo) e percorre pela direita até chegar ao final da linha, continuando na linha inferior. O primeiro ponto marcado como forma pincelada que for encontrado (que tiver valor 1 na matriz), será considerado o primeiro ponto detectado como borda.

A partir deste ponto, analisa-se todos os vizinhos adjacentes na procura pelo próximo. Cria-se um *grid*, com este ponto ao centro, e percorre-se em sentido horário, a partir da posição central esquerda, até encontrar outro ponto pertencente a borda. Quando isto ocorrer, este novo ponto será o centro do *grid*. A procura continuará da mesma forma, em sentido horário, começando pela posição anterior no *grid* ao último ponto encontrado. (Figura 17)

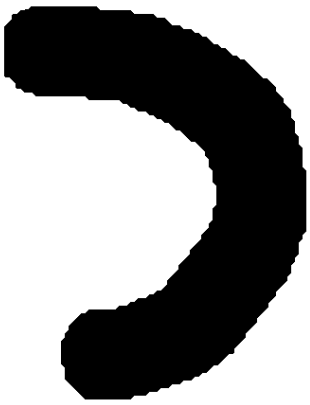

Figura 16: Exemplo: imagem de uma pincelada feita pelo usuário. (tamanho original 128x128 *pixels*)

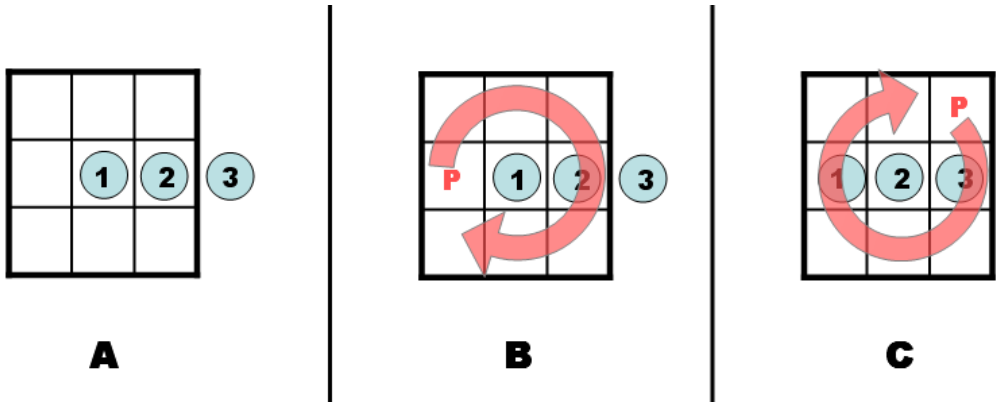

Figura 17: Exemplo: A) Alguns pontos da borda na forma pincelada. B) *Grid* criado com o primeiro ponto ao centro. Busca pelo próximo ponto em sentido horário, começando pela posição central à esquerda no *grid* (posição de procura representada pelo "P" na figura). C) Com o segundo ponto detectado, ele será o novo centro do *grid*. A busca continua a partir da posição anterior ao último ponto encontrado.

```
begin
    pos ← 1
    while ultimo_ponto_detectado ! = primeiro_ponto do
        //Considerando i, t como x, y do último ponto detectado, determina-se o
          local de procura em sentido horário no grid de 8 posições
        switch pos_proc do
            case 1
                procura_x ← (i)
                procura_y ← (t − 1)
            end
            case 2
                procura_x ← (i + 1)
                procura_y ← (t − 1)
            end
            case 3
                procura_x ← (i + 1)
                procura_y ← (t)
            end
            //Segue da mesma forma em sentido horário até posição 8
            .
            .
            .
        end
        if matriz[procura_x][procura_y] == 1 then
            //Atualiza posição da borda atual
            i ← procura_x
            t ← procura_y
            //Próxima posição de procura é a anterior do último ponto encontrado
            //Se estava-se na posição de procura 1, anterior será 8
            if pos_proc == 1 then
                pos_proc ← 8
            else
                pos_proc − −
            end
            pos + +
            borda_x[pos] ← procura_x
            borda_y[pos] ← procura_y
        else
            //Se não for borda, segue procurando em sentido horário no grid
            pos_proc + +
            //Se ao incrementar posição de procura chega-se a 9, próxima será 1
            if pos_proc == 9 then
                pos_proc ← 1
            end
        end
    end
end
```

Algorithm 2: Detecção dos demais pontos da borda

Este processo é repetido até o primeiro ponto da borda ser encontrado novamente, fechando então a forma e finalizando a detecção de todos os pontos que determinam a borda. (Figura 18)

Figura 18: Exemplo: borda detectada (pontos vermelhos para ilustração) da forma exibida na Figura 16.

3.2.1.2 Geração da área de extrude simulado $< ae >$

A partir da detecção da borda, segue-se para o próximo passo, ou seja a determinação de vetores que representam a direção de expansão da forma pintada, assim como as cores com que esta será preenchida, para a geração do *normal map*.

Para cada dois *pixels* vizinhos na borda $< P1, P2 >$, conforme ilustrado na Figura 19, é determinado o vetor entre eles. Após, encontra-se o vetor perpendicular $< v1 >$, que aponta para fora do interior da forma. A este vetor será adicionado uma distância $< d >$ relacionada ao atributo de "área de *extrude* simulado" $< ae >$ (citado anteriormente), e então teremos o ponto $< P3 >$ que juntamente com outros pontos obtidos da mesma forma, definem $< ae >$.

Este processo se repete em cada ponto da borda, até termos uma cadeia de pontos externos à forma da pincelada que delimita a área de inclinação das normais. Então a cor a ser pintada no *normal map* nesta área, será determinada pelo vetor médio entre os vetores perpendiculares, a cada dois vizinhos na borda. (Figura 20)

Scanline

O último passo a determinar para criar o *normal map* é a área exata na qual será pintada cada cor diferente. Esta área é determinada pela execução de um algoritmo de *scanline*, que detecta pontos de intersecção entre vários vetores (polígonos, ou segmentos de reta). Neste caso, os vetores são gerados por:

- Os dois pontos vizinhos na borda que estão sendo analisados no momento ($< B1 >$, $< B2 >$);

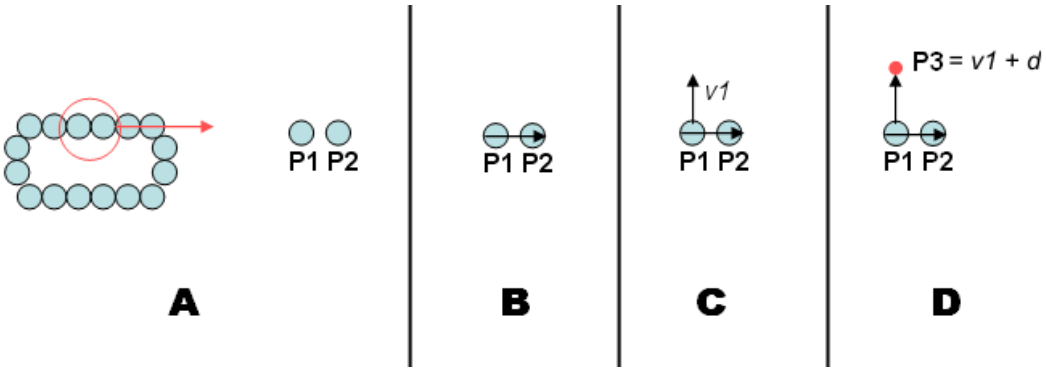

Figura 19: A) Exemplo: dois pontos da borda. B) Determinação do vetor entre eles. C) Determinação do vetor perpendicular $< v1 >$. D) Somando a distância de *extrude* desejado $< d >$, temos o ponto final referente a área máxima de alteração de normais a este *pixel* ($< P3 >$ na figura).

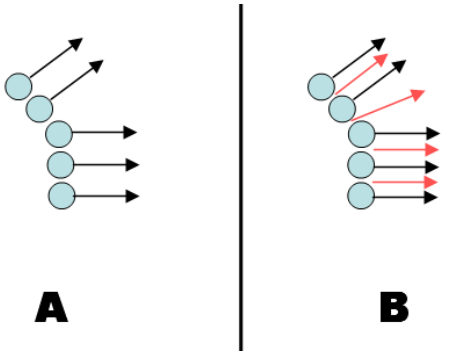

Figura 20: A) Exemplo: alguns pontos da borda com seus respectivos vetores perpendiculares. B) Vetores médios (em vermelho) que determinam a cor a ser preenchida no *normal map*.

- Um destes pontos $< B1 >$ e seu respectivo ponto externo $< P1 >$ que delimita $< ae >$ como comentado na seção 3.2.1;
- Outro ponto $< B2 >$ e seu respectivo ponto externo $< P2 >$;
- Os dois pontos externos $< P1 >, < P2 >$.

Os pontos de intersecção encontrados entre todos estes vetores, para cada iteração do algoritmo de *scanline*, determina o intervalo em que devem ser pintadas as cores no *normal map*. A Figura 21 exemplifica cada passo deste procedimento.

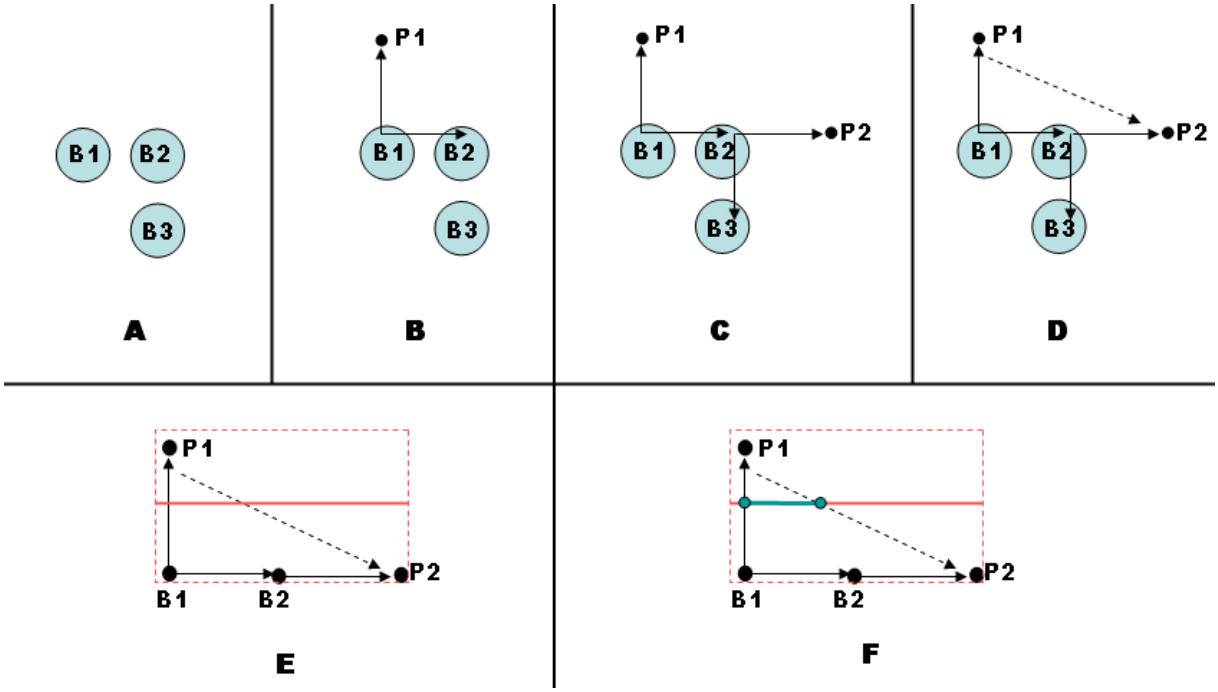

Figura 21: A) Exemplo: três pontos da borda. B) Determinação do vetor entre eles e vetor perpendicular, definindo P1. C) Mesmo procedimento referente ao próximo ponto da borda (B2 na ilustração). D) Criação do vetor entre P1 e P2. E) Scanline (linha vermelha) entre os vetores B1-B2; B1-P1; P1-P2; B2-P2. F) Será pintado as cores no intervalo entre os pontos de intersecção encontrados (linha verde).

3.2.1.3 Suavização

Dependendo da resolução do *normal map* a ser criado, e da forma pincelada pelo usuário, as normais das bordas podem gerar, em certas áreas, transições não suaves nas cores do *normal map*. Isto resulta num revelo mais irregular, diminuindo a qualidade do resultado final no objeto 3D. Para resolver o problema, é possível realizar um processamento extra que busca suavizar os pontos que delimitam as bordas da $< ae >$, deixando-as com transições de cores mais suaves. Esta suavização é uma média ponderada entre a posição atual do ponto (com peso = 2) e os pontos vizinhos (anterior e posterior, com pesos = 1). Resultados da suavização podem ser vistos na Figura 35 (no próximo capítulo).

3.2.1.4 Tratamento de quinas

O procedimento de detecção da borda da forma pincelada e a determinação dos vetores, como descrito anteriormente, pode gerar algumas situações não desejáveis se for utilizado um pincel de formato quadrado, ao desenhar as formas. Pois neste caso têm-se bordas quadradas na forma pintada. Assim, nas quinas desta forma, o sistema de *scanline* iria pintar uma área maior que a desejada, pois o ângulo reto no desenho da pincelada causaria este efeito. A parte superior da Figura 22 (letras A,B,C), ilustra o problema comentado.

Para resolver este problema, definiu-se um tratamento especial para as quinas, da seguinte forma: utiliza-se o ângulo formado pelos vetores normais a borda da pincelada ($< V1 > e < V2 >$ na Figura 22(A)), a cada dois *pixels* vizinhos da borda, por exemplo $< B1 > e < B2 >$. Se este ângulo for inferior a um valor pré-estabelecido, considera-se estes pontos como sendo "quina" e então aplica-se um tratamento específico. Este, consiste em copiar o vetor perpendicular do ponto anterior $< V1 >$, aplicando-o ao próximo ponto $< B2 >$, de forma que a área da *scanline* nesta quina fique coerente com o desejado. O valor do ângulo que determinará se o ponto será considerado como quina pode ser alterado pelo usuário, mas o valor padrão já delimita que ângulos retos sejam considerados como quina, fazendo com que formas retangulares gerem *normal maps* corretos. A Figura 22 demonstra o problema e a correção utilizando o sistema de quina.

Opcionalmente, este sistema de tratamento de quinas pode ser evitado, atingindo-se um resultado diferente no normal map. Para isso, os vetores perpendiculares dos pontos vizinhos nas quinas da forma retangular, que criam o problema comentado, podem ser gerados usando-se gradiente. Assim, os vetores seriam inclinados, formando quinas exatamente iguais ao processo tradicional de desenvolvimento de *normal maps* (a partir de uma segunda malha geométrica de referência). Então o uso de gradiente pode ser escolhido pelo usuário, de acordo com o desenho de quina que preferir em formas retangulares. A figura 23 demonstra o efeito desta opção.

3.3 Modo animação

O modelo visa explorar animações de *normal map*, idéia ainda não abordada na literatura, ao que se conhece. Neste caso, o protótipo poderá gerar vários *normal maps* em função do tempo, que seriam os quadros de uma animação. Carregando essa seqüência de quadros, a determinadas taxas de exibição, em um aplicativo em tempo real, como jogos, teria-se a animação do relevo no objeto.

Para gerar animações criadas manualmente pelo usuário, são utilizados *keyframes*, que são os quadros chaves de uma animação. Assim, basta manipular o *normal map* na forma desejada e marcar como um *keyframe* (quadro chave). Para isso há uma linha do tempo, com vários quadros que compõem o tempo. A velocidade normal de uma animação, para se ter boa sensação de movimento na tela do computador, é de 30 quadros por segundo. Podem ser usados menos quadros, porém isto geralmente degrada a fluidez da animação.

No espaço de tempo entre os *keyframes* especificados, serão geradas variações dos

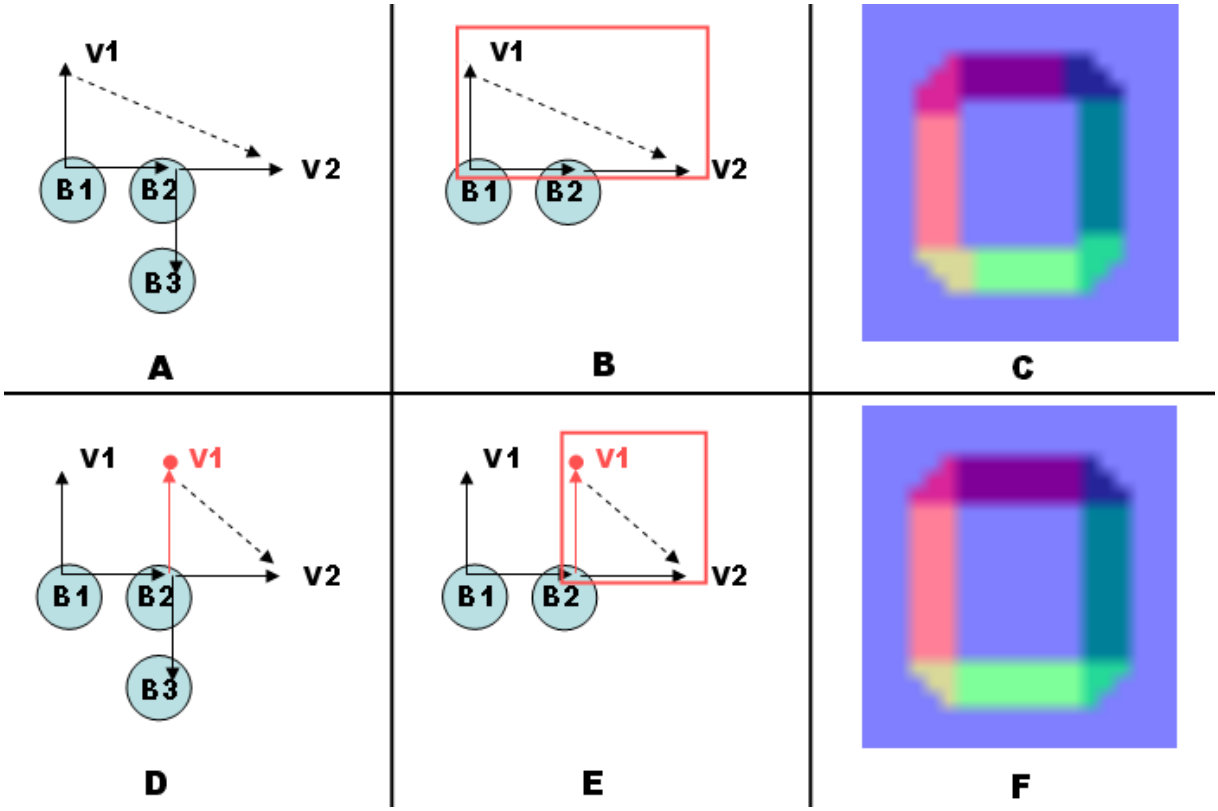

Figura 22: A) Pontos da borda(B1,B2,B3), representando uma quina. Vetores perpendiculares definem V1,V2. B) Área da scanline (retângulo vermelho). C) *Normal map* resultante, com desenho incorreto nas quinas da forma. D) Mesma situação de A, porém considerando estes pontos como quina, duplicando vetor perpendicular V1, e aplicando em B2. E) Área da scanline reduzida. F) *Normal map* resultante, com desenho correto nas quinas da forma.

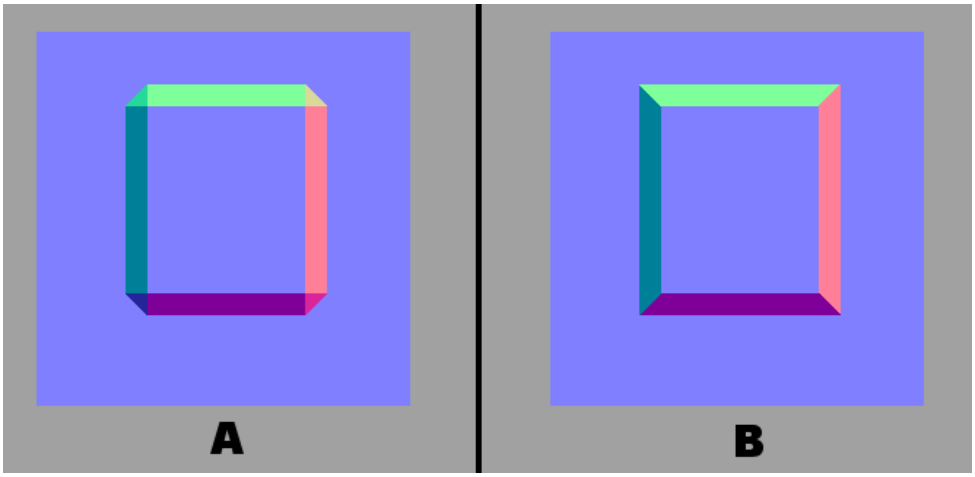

Figura 23: A) Sem utilizar gradiente. B) Utilizando gradiente (quinas de formas retangulares apresentam resultado idêntico ao processo tradicional de geração de *normal maps*).

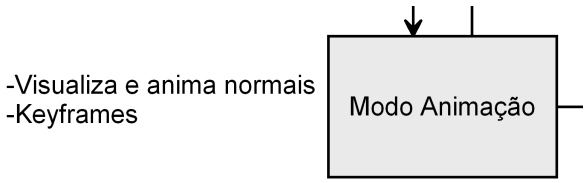

Figura 24: Parte do modelo referente a manipulação das normais com animação.

vetores normais, compondo a animação, através de simples interpolação linear. Este sistema de *keyframe* está em conformidade com o usual em $software$s comerciais.

3.4 Saída *normal map*

A última parte do modelo trata da exportação do *normal map* final, resultante das alterações realizadas pelo protótipo. Este é o principal objetivo deste trabalho: gerar um *normal map* que possa ser usado na aplicação externa desejada, como um jogo.

No caso da animação, serão exportados vários *normal maps*, um para cada quadro de tempo, de acordo com a configuração da animação. Assim todos os mapas poderão ser carregados na aplicação destino, recriando a animação no relevo representado pelo *normal map* no objeto 3D.

4 PROTÓTIPO

Neste capítulo, são comentados os detalhes da implementação do protótipo desenvolvido.

Este protótipo tem por objetivo rodar em um ambiente que permita a visualização de um modelo poligonal em 3D, com o *normal map* aplicado. Além disso, a possibilidade de realizar modelagens geométricas, editando vértices e faces, é útil uma vez integrada ao mesmo ambiente. Por isso foi escolhido utilizar uma aplicação de modelagem e animação comercial, aproveitando todos estes recursos existentes e focando na programação das novas ferramentas que executassem os procedimentos descritos no modelo deste trabalho. Assim o protótipo implementado foi escrito em MEL (Maya Embedded Language), que é a linguagem de scripts interna do *software* 3D Maya (este *software* foi comentado no capítulo 2, seção 2.3).

Outra vantagem em relação a utilização do Maya, é aproveitar sua interface chamada Artisan (Figura 25), composta de um sistema que simula pintura no modelo poligonal. Este recurso é usado com vários propósitos no *software*, como pintar cores na textura de um modelo 3D, pintar diversos atributos como direção de cabelos no módulo de simulação de cabelos dinâmicos, influência de vértices em relação a um esqueleto usado para deformar personagens, dentre outras funções.

O Maya ainda possui uma ferramenta chamada "Paint Scripts Tool", que utiliza a interface Artisan para aplicar scripts feitos pelo usuário, em certas partes de um modelo poligonal. Então o protótipo foi programado escrevendo-se um script para ser usado com esta ferramenta. Assim a base da interface, como permitir a pintura em geometria 3D, configurando o tamanho e forma do pincel, foi disponibilizada pelo Maya nativamente. Já a determinação das funções aplicadas a cada pincelada do usuário com esta ferramenta, foi escrita em script.

O protótipo deste trabalho necessita criar os *normal maps* resultantes do processamento da "pintura", pois esta textura deve ser aplicada no objeto e visualizada na janela interativa do Maya. Porém esta função de criar um arquivo de imagem em disco, determinando por código para cada cor de cada *pixel*, não é suportada diretamente pela linguagem de scripts MEL. Desta forma, foi necessário desenvolver um *plug-in* para Maya, que constitui-se em um *software* escrito na linguagem C++, utilizando a API (Application Program Interface) do Maya. Este *plug-in* permite então escrever uma imagem em disco determinando diretamente todas as cores, havendo uma comunicação entre o script para informar estes dados.

Assim o protótipo deste trabalho é composto de um script, que é utilizado com a interface Artisan do Maya, que se comunica com um *plug-in* para a criação do *normal*

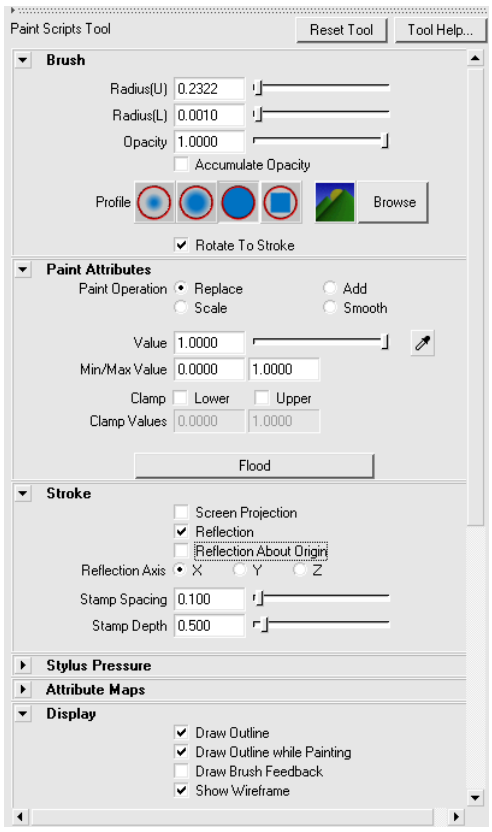

Figura 25: Interface artisan do Maya.

map em disco. A Figura 26 demonstra a janela principal do script.

Para executar o protótipo, após ter instalado o script e *plug-in* no Maya, o usuário necessita de um modelo poligonal para servir como base da pintura. Este modelo pode ser criado usando as ferramentas tradicionais de modelagem do Maya, ou importado de outro $software$ usando um formato de arquivo compatível. Então com este objeto selecionado, executando o "Paint Scripts Tool" devidamente configurado para rodar o script, a interface principal surgirá (Figura 26), permitindo realizar a pintura sobre o objeto. A cada pincelada do usuário, o *normal map* correspondente será criado e aplicado no objeto 3D, permitindo a visualização do efeito no mesmo. Este procedimento obedecerá as configurações do script, que serão descritas abaixo, juntamente com cada função que poderá ser acionada pelo usuário.

4.1 Configuração do *normal map* e "extrude" simulado

Nesta primeira parte do script, demonstrada pela Figura 27, temos a configuração do *normal map* que será criado pelo protótipo (definido por $< nmi >$ no capítulo anterior). Pode-se configurar o tamanho do arquivo a ser criado, sua altura e largura em *pixels*. No atributo "Largura do extrude simulado" configuramos a distância máxima da borda da pincelada que será usada para inclinar as normais, equivalente ao tamanho da área

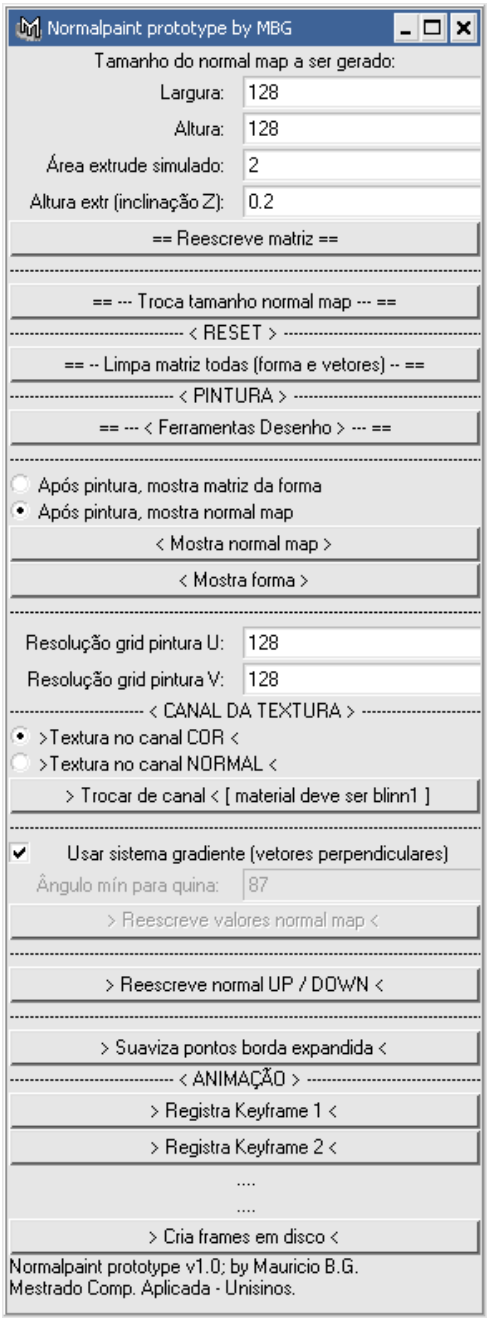

Figura 26: Janela principal do protótipo escrito em MEL.

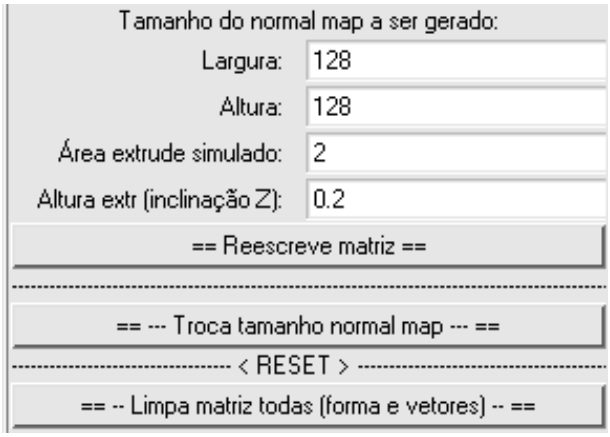

Figura 27: Primeira parte da interface do script.

colorida ao redor da forma central no *normal map*, como descrito no capítulo 3, seção 3.2 (exemplo na Figura 14).

Como os vetores normais serão inclinados somente nos eixos X e Y de acordo com a pincelada, o valor da inclinação no eixo Z pode ser definido pelo usuário. O atributo "Altura extrude (inclinação Z)", que está com o nome abreviado na interface, controla este parâmetro. A inclinação pode conter valores de -1 a 1. O resultado influencia nos tons das cores no *normal map*, representando um relevo mais alto (brusco) ou plano (suave).

O botão "Reescreve matriz", permite ao usuário perceber a diferença da troca de valor do atributo "Largura do extrude simulado" e "Altura extrude (inclinação Z)", pois o script recalcula a pincelada de acordo com os novos valores. É possível também limpar todas as matrizes do script que guardam as informações da forma "pintada" pelo usuário, bem como o *normal map* gerado a partir dela. O botão "Limpa matriz todas (forma e vetores)" realiza esta operação. Então temos a textura "limpa", equivalente a apagar todas as formas "pintadas" e recomeçar do zero.

Quando se desejar trocar o tamanho do *normal map* a ser criado pelo protótipo, após alterar os atributos de largura e altura descritos acima, deve-se acionar o botão "Troca tamanho normal map", para os novos valores serem aplicados.

4.2 Ferramentas de pintura

Na Figura 28 é exibida a janela referente a pintura no script. Ela pode ser exibida acionando-se o botão "Ferramentas Desenho", na janela principal. Para pintar normalmente, atualizando o *normal map*, nenhum comando ou botão é necessário, bastando o script estar acionado. Neste caso somente uma pincelada deve ser feita, e os botões nesta janela permitem continuar pintando de acordo com o desejo do usuário, fazendo novas formas a partir da atual ou independente dela.

O botão "Nova pincelada" permite que o usuário execute uma nova pintura, mas sem apagar o que já foi pintado anteriormente. Pode-se então gerar um *normal map* a partir de várias pinceladas diferentes. Já o botão "Continua pincelada" permite criar

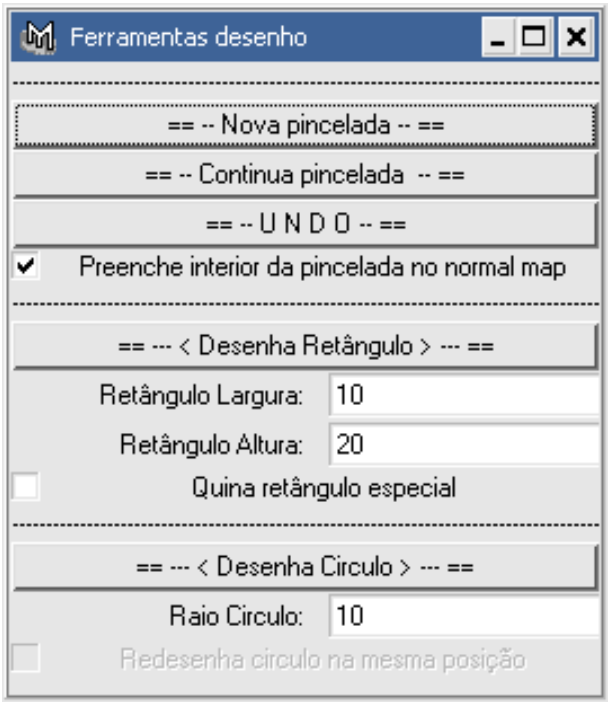

Figura 28: Segunda janela da interface do script.

uma forma mais complexa, pintando a partir do traçado já existente, continuando a forma. Para desenhar a letra "T", por exemplo, basta executar duas pinceladas que se conectam. Isto é feito desenhando a primeira delas, então acionando o botão que informará ao script que esta pincelada será continuada e após executando a segunda.

O comando "UNDO" permite apagar a última pincelada, retornando à cor padrão do normal map onde ela existia. O checkbox "Preenche interior da pincelada no normal map", faz com que o interior de uma nova pincelada seja preenchida com a cor padrão do *normal map* para não conflitar com alguma forma já existente. A Figura 29 demonstra o efeito da opção ligada e desligada ao pintar.

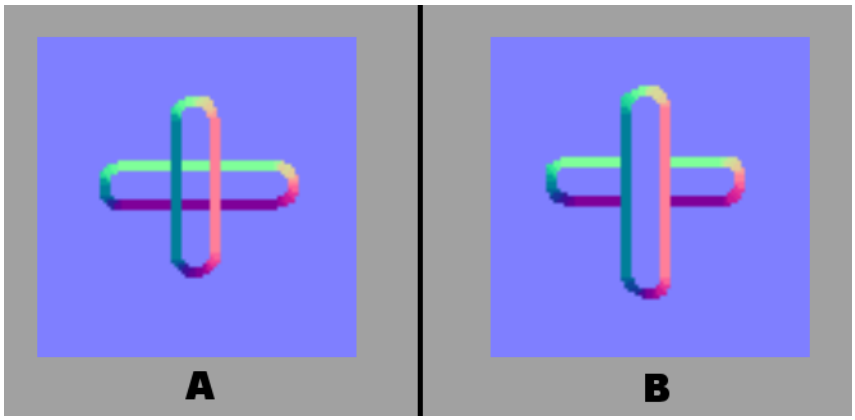

Figura 29: Exemplo do efeito da opção "Preenche interior da pincelada no normal map" entre duas pinceladas diferentes. A) Opção desligada. B) Opção ligada.

Foram implementados alguns facilitadores para se desenhar formas mais regulares. Há um botão para desenhar retângulos e outro para círculos. O usuário pinta uma forma qualquer em determinada área do objeto 3D e pode transformar esta forma em retângulo ou círculo acionando o respectivo botão. Este procedimento leva em consideração o primeiro ponto detectado na borda da forma pintada (primeiro ponto na extremidade superior esquerda da forma) e torna este ponto a coordenada central do círculo, ou a extremidade superior esquerda do retângulo. Os parâmetros no script, de largura e altura do retângulo e raio do círculo, podem ser ajustados e atualizados acionando novamente o botão de desenho da forma. O checkbox "Quina retângulo especial", deve ser acionado juntamente com a opção de usar o sistema de gradiente para geração dos vetores perpendiculares (opção na janela principal do script), se o usuário quiser *normal maps* com quinas retangulares exatamente iguais ao processo de geração tradicional. Esta opção diz ao script que o desenho das cores deste respeitar as quinas inclinadas resultantes do uso de gradiente.

O último checkbox desta janela do script, "Redesenha círculo na mesma posição" (que é desabilitado por padrão), permite que novos círculos desenhados sejam criados na mesma posição do anterior, ou seja, o usuário mantém esta opção ligada para modificar o raio de um círculo existente e desligada para desenhar novo círculo em outra posição.

4.3 Configuração de exibição e *grid*

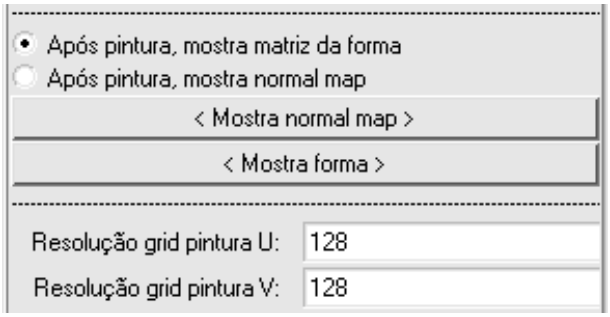

Figura 30: Segunda parte da interface na janela principal do script.

É possível gerar e aplicar, no objeto 3D, uma textura que demonstra a forma central da pincelada do usuário, ao invés do *normal map*. Este recurso é útil para referência visual somente, demonstrando não só a forma central, como também realçando a borda detectada (pontos vermelhos) e a área máxima da inclinação das normais (pontos verdes), referente ao atributo "Largura do extrude simulado". (Figura 31)

Pode-se configurar se após a pintura será mostrada a imagem da forma como descrito acima, ou o *normal map* resultante dela. Além disso há dois botões que executam esta função no momento que forem acionados, podendo alternar entre a forma e o *normal map* no instante desejado, observando a mudança referente as pinceladas já realizadas.

O *grid* que representa a discretização da face do objeto, permitindo a leitura da posição correta do pincel na face, pode ter sua resolução alterada. Quanto maior for a

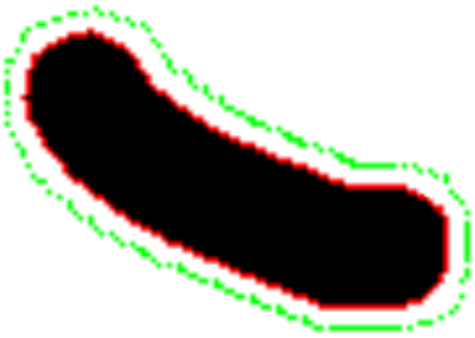

Figura 31: Exemplo de forma pincelada pelo usuário. Pontos pretos são a forma em si, pontos vermelhos representam a borda detectada e pontos verdes representam a área máxima da inclinação das normais.

resolução, mais suave será a forma pintada, mas também será necessário maior processamento. A resolução pode ser alterada trocando-se os valores dos atributos "Resolução grid pintura U" e "Resolução grid pintura V", onde U e V referem-se à horizontal e vertical, no espaço bidimensional da textura.

4.4 Canais, sistema de quina, direção do relevo e suavização

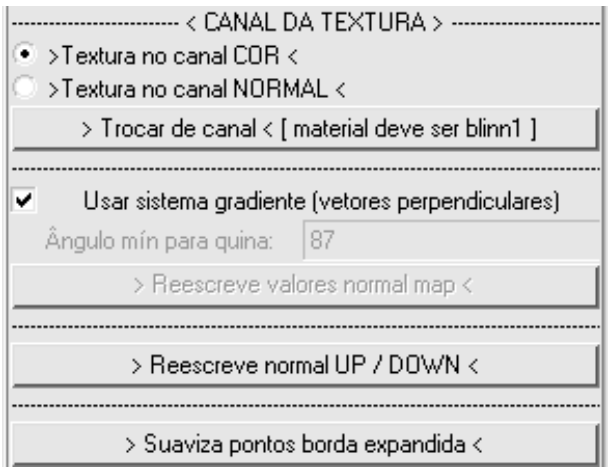

Figura 32: Terceira parte da interface do script.

A função principal do protótipo é criar o *normal map* e aplicar ao objeto 3D em questão, como dito anteriormente. Mas na realidade, em um *software* de modelagem 3D como o Maya, aplica-se texturas à "materiais" ou *shaders*, que por sua vez são aplicados

ao objeto 3D. Este "material" contém todas as configurações sobre a representação visual do objeto, como sua reflexão com a luz, cor principal, transparência, entre vários outros atributos organizados em canais, permitindo ainda aplicar texturas nestes canais. Para surgir o efeito de relevo simulado característico do *normal map*, este deve ser aplicado no canal respectivo do "material" que está sendo usado pelo objeto. Porém, pode ser interessante observar as cores do *normal map*, como referência visual, ao invés de seu efeito. Para isto o protótipo possibilita determinar se a textura criada será aplicada no canal de cor, ou no canal de alteração das normais, dentro do "material". Então pode-se alternar entre ambos para ter o efeito de relevo do *normal map* ativo, ou somente observar suas cores como se fosse uma textura de cor tradicional. A Figura 33 demonstra o efeito da troca de canais.

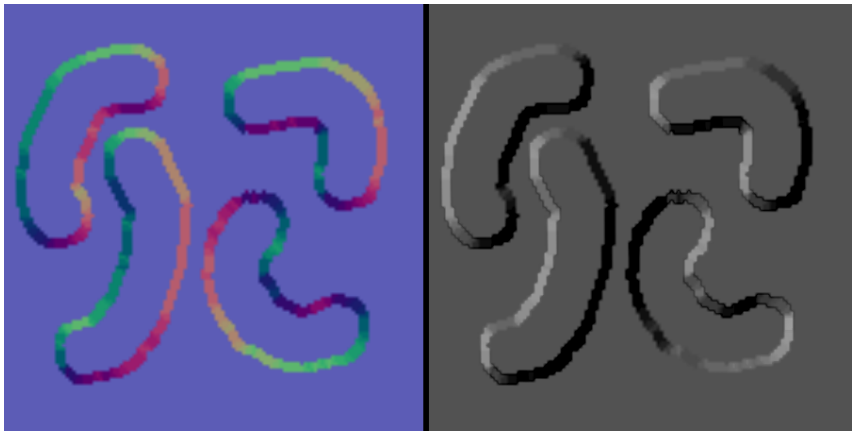

Figura 33: Exemplo da troca de canais no "material". Na esquerda a textura está aplicada no canal de cor, já na direita no canal de alteração de normais. (resolução original das texturas: 128x128 *pixels*)

Como comentado no capítulo 3, seção 3.2, se for utilizado um pincel de formato quadrado, o protótipo realiza um tratamento especial para as quinas. As quinas são detectadas através de um valor pré-especificado pelo usuário alterando o atributo "Ângulo mín para quina" na interface do script. O botão "Reescreve valores normal map" permite recalcular e redesenhar o *normal map* para visualização do efeito que a alteração do valor realizou. Estas opções estão desabilitadas por padrão, serão utilizadas somente se o checkbox "Usar sistema gradiente (vetores perpendiculares)" for desmarcado. Se este continuar selecionado, o sistema de gradiente será utilizado e demais tratamentos especiais de quinas não serão necessários.

O usuário pode desejar criar um relevo que represente elevação ou depressão. Por isso o protótipo possui em sua interface o botão "Reescreve normal UP/DOWM", que faz o relevo criado pela última pincelada ter sua direção revertida. Isto ocorre invertendo a direção de todos os vetores que determinam as cores que serão pintadas no *normal map*. A Figura 34 mostra um exemplo deste recurso.

Outra possibilidade que o protótipo permite, é a suavização dos pontos gerados pela expansão da forma pincelada, como comentado no modelo deste trabalho (capítulo 3). Seu objetivo é suavizar também as cores no *normal map* criado, gerando uma transição mais homogênea entre as cores. Esta suavização é realizada através de uma média ponderada entre a posição de um ponto expandido da borda, com seus respectivos vizinhos anterior

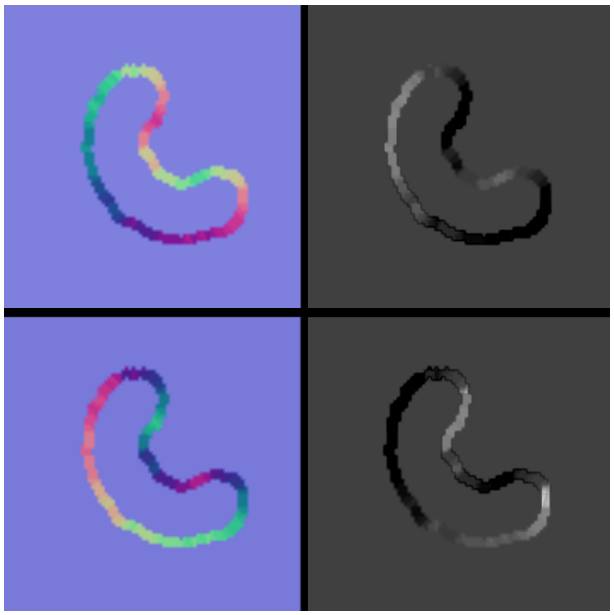

Figura 34: As imagens superiores representam o *normal map* padrão, e seu respectivo relevo convexo. As imagens inferiores representam a inversão da direção do *normal map*, e seu respectivo relevo côncavo.

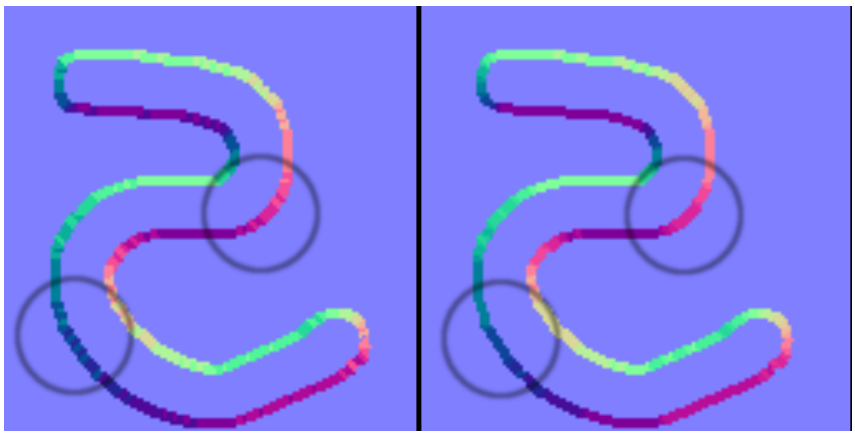

Figura 35: Exemplo de suavização. A imagem da direita contém o *normal map* suavizado (círculos ilustrativos para ressaltar áreas com as cores suavizadas).

e posterior. O cálculo é feito usando-se a média do dobro da posição do ponto adicionado aos seus vizinhos. Assim têm-se uma melhor disposição dos mesmos, porém sem perder a referência das posições originais (Figura 35).

4.5 Animação

Figura 36: Quarta parte da interface do script.

Quanto a animação, o protótipo permite a demarcação de *keyframes* na linha de tempo existente no Maya. O usuário escolhe um frame da linha do tempo, altera o *normal map* da forma que quiser e o marca como sendo o primeiro *keyframe*, acionando o botão respectivo na interface do script. Após, seleciona outra posição no tempo, altera o *normal map* novamente e marca o segundo *keyframe* (Interface do script referente à animação: Figura 36).

Então para realizar a animação, salvando a seqüencia de *normal maps* em disco, aciona-se o botão "Cria frames em disco". O protótipo irá gerar as texturas utilizando interpolação linear entre a direção de todos os vetores normais que compõem os *normal maps* demarcados como *keyframes*. Isto ocorrerá no intervalo de tempo entre estes *keyframes*. O script possibilita também carregar cada textura no quadro de tempo respectivo, permitindo que o usuário rode a animação assistindo, na janela interativa do Maya, a atualização da textura aplicada ao objeto.

5 RESULTADOS

Neste capítulo, são exibidos os resultados obtidos com a utilização do protótipo.

Como comentado anteriormente, o principal objetivo deste trabalho é permitir ao usuário criar, pelo sistema de pintura, o *normal map* com qualidade equivalente ao realizado pelo processo tradicional em que se utiliza um segundo modelo geométrico como referência.

Uma forma de validar este novo método de pintura de *normal maps* é apresentar modelos e texturas criadas da forma tradicional e da nova forma, comparando ambos. Para tanto, optou-se por prover a comparação utilizando-se métricas de qualidade visual e tempo necessário de utilização dos sistemas para realizá-los. Para os exemplos apresentados neste capítulo, a resolução dos *normal maps* gerados são de 128x128 *pixels*.

Um simples relevo quadrado aplicado à um plano poligonal, para ser criado pela forma tradicional, requer que o usuário crie o plano poligonal que será aplicada a textura e uma cópia dele que será subdividida. Então seleciona-se a face frontal deste segundo plano e aciona-se a ferramenta *"extrude face"* que permite criar novas faces. Com ela será feita a modelagem, que neste caso é muito simples, basta redimensionar a nova face, aplicar novamente o *"extrude face"*, ajustando a recém criada face formando o relevo final. Desta forma, com o modelo original e o de referência prontos, deve-se acionar a ferramenta *"surface sampler"* do Maya, que fará a análise dos dois objetos 3D, criando em disco o *normal map* final. Se houver necessidade de alterar o *normal map* para algum ajuste, é necessário modificar a geometria do modelo 3D de referência, e executar novamente o *"surface sampler"*.

Para aplicar este *normal map* ao objeto simplificado, observando o efeito de relevo por ele representado na janela interativa do Maya, é necessário criar um material (que guarda todos os atributos visuais do modelo 3D) e o atribuir ao objeto. Neste material deve ser carregado a textura do *normal map* no canal respectivo. Então finalmente acionando a opção de exibir objetos com alta qualidade visual na janela interativa do Maya, pode-se notar o efeito de relevo, que reage em tempo real à iluminação da cena.

No caso do novo sistema de pintura apresentado neste trabalho, para recriar este mesmo relevo simples, basta criar o modelo 3D simplificado em que será aplicado o *normal map* e pintar sobre ele utilizando o protótipo. Para criar um relevo quadrado na posição da pincelada, pode-se usar a ferramenta do protótipo para este fim, determinando altura e largura do mesmo. Em alguns segundos o *normal map* será criado e aplicado ao objeto 3D automaticamente. Para realizar-se qualquer alteração desejada ao relevo, basta pintar novamente que o *normal map* será atualizado.

As Figuras 37 e 38 demonstram os resultados obtidos pelo procedimento tradicional

e pelo novo sistema de pintura, juntamente com os tempos aproximados da execução de cada um. Vale ressaltar que o tempo de modelagem de geometria pode variar em relação a experiência e velocidade de trabalho do artista 3D. Os valores apresentados nos exemplos devem ser considerados somente como uma referência estimada. Além disso, as diferenças entre as extremidades da forma nos *normal maps* das figuras acima citadas, deve-se a uma opção de implementação do modelo, que prioriza formas mais orgânicas, conforme discutido no capítulo 4.

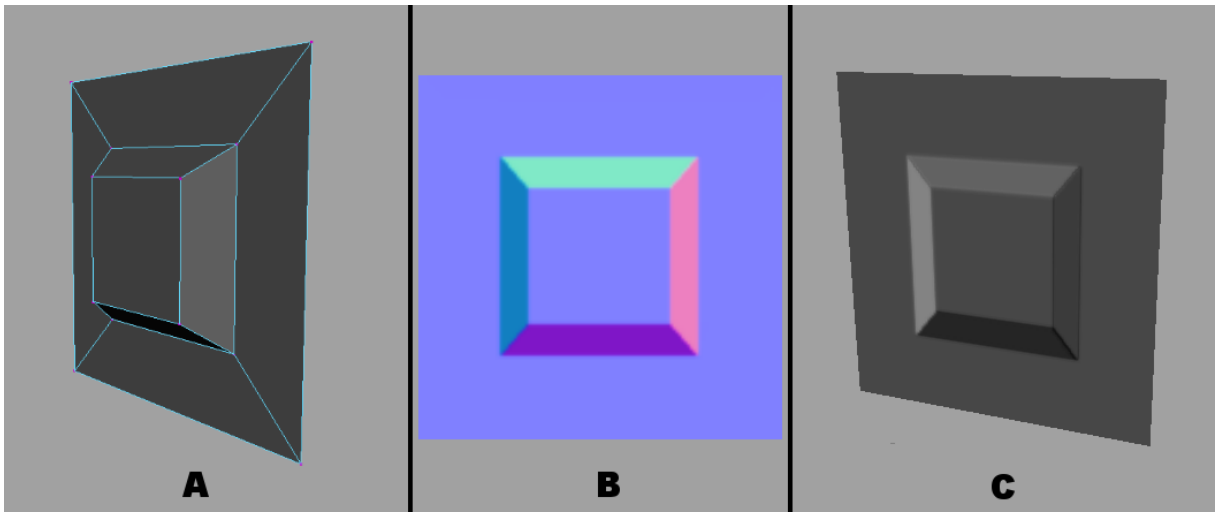

Figura 37: Geração da textura pelo processo tradicional: A) Modelagem de referência. B) *Normal map* criado. C) *Normal map* aplicado ao plano poligonal demonstrando o relevo. (Tempo total estimado do procedimento: 1 minuto.)

Já para uma forma mais orgânica, o trabalho de modelagem tradicional pode ser bem mais complexo e demorado. Quanto mais faces forem necessárias ao modelo 3D de referência, mais trabalho o artista 3D terá em criá-las e ajustá-las. Como exemplo, um relevo que represente a letra grega π, exigirá a criação de primitivas 3D que serão subdivididas e manipuladas até determinar a forma desejada. A Figura 39 demonstra os passos necessários neste processo. Para este mesmo exemplo, o novo sistema de pintura poderá recriar a forma com algumas pinceladas. A Figura 40 demonstra o resultado.

Objetos 3D, utilizados em jogos ou outras aplicações interativas, costumam utilizar várias texturas para compor o visual desejado. Além do *normal map* com objetivo de criar a ilusão de relevo, são utilizados texturas aplicadas ao canal de cor principal, misturando então o visual das cores com o relevo. O exemplo da Figura 41 ilustra este caso, onde um objeto simbolizando uma porta foi criado. Para ele foi desenvolvido uma textura de cor tradicional, além do *normal map*. A Figura 42 ilustra a recriação do relevo utilizando o protótipo.

A Figura 43 mostra um objeto que representa uma casa simples. A modelagem de referência foi desenvolvida a partir de um objeto mais complexo que um plano poligonal. Nestes casos, as coordenadas de mapeamento devem ser manipuladas para que todas as faces do objeto leiam corretamente a textura dentro do espaço bidimensional de textura (UV). O *normal map* sempre é criado baseado nestas coordenadas, fazendo com que encaixe perfeitamente com a textura aplicada no canal de cor principal. Além disso as

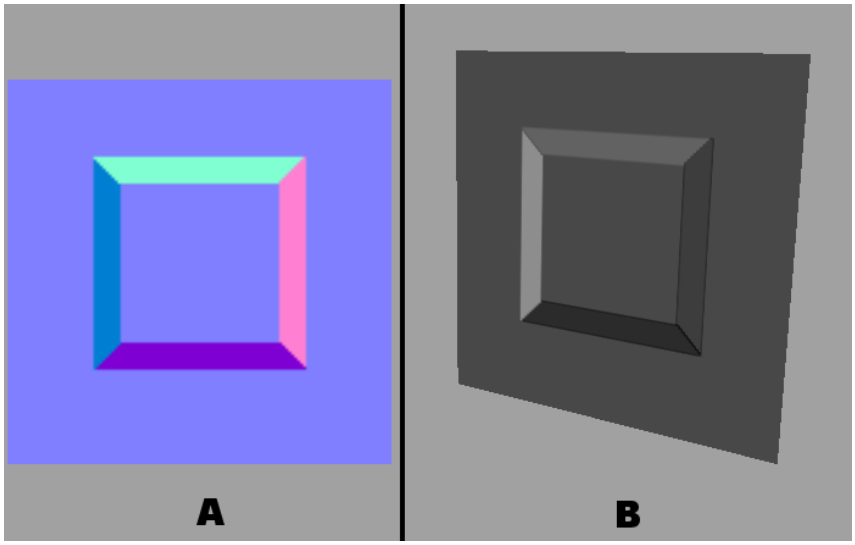

Figura 38: Geração da textura pelo novo sistema de pintura: A) *Normal map* criado. B) *Normal map* aplicado ao plano poligonal demonstrando o relevo. (Tempo total estimado do procedimento: 25 segundos.)

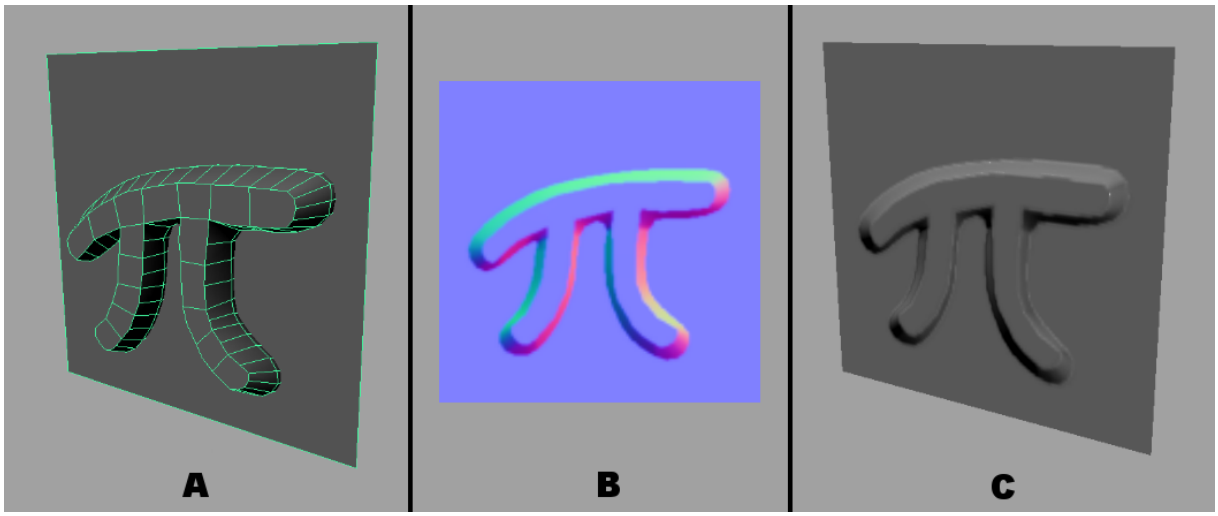

Figura 39: Geração da textura pelo processo tradicional: A) Modelagem de referência. B) *Normal map* criado. C) *Normal map* aplicado ao plano poligonal demonstrando o relevo. (Tempo total estimado do procedimento: 18 minutos.)

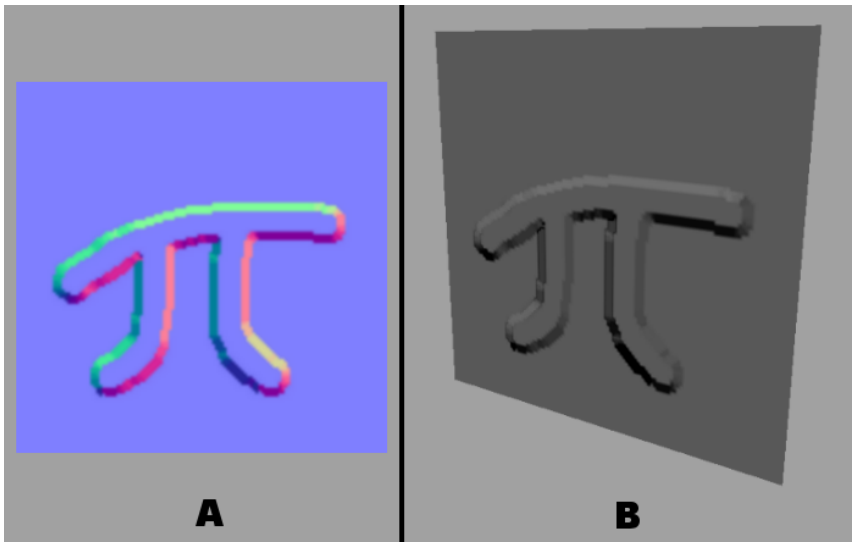

Figura 40: Geração da textura pelo novo sistema de pintura: A) *Normal map* criado. B) *Normal map* aplicado ao plano poligonal demonstrando o relevo. (Tempo total estimado do procedimento: 1 minuto e 45 segundos.)

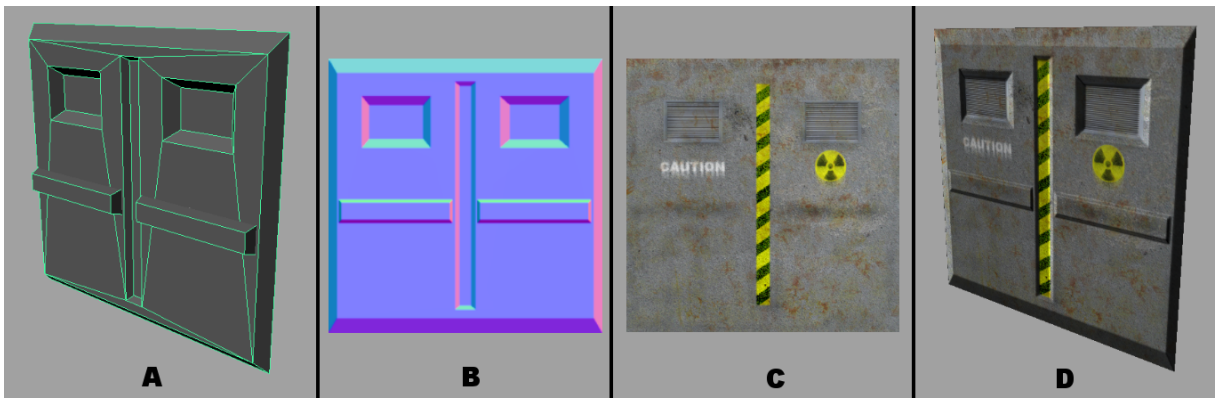

Figura 41: Geração da textura pelo processo tradicional: A) Modelagem de referência. B) *Normal map* criado. C) Textura desenvolvida em um editor de imagens, para ser aplicada ao canal de cor no objeto 3D. D) Texturas aplicadas ao plano poligonal. (Tempo total estimado do procedimento: 4 minutos e 10 segundos. Este tempo não considera o desenvolvimento da textura extra aplicada ao canal de cor.)

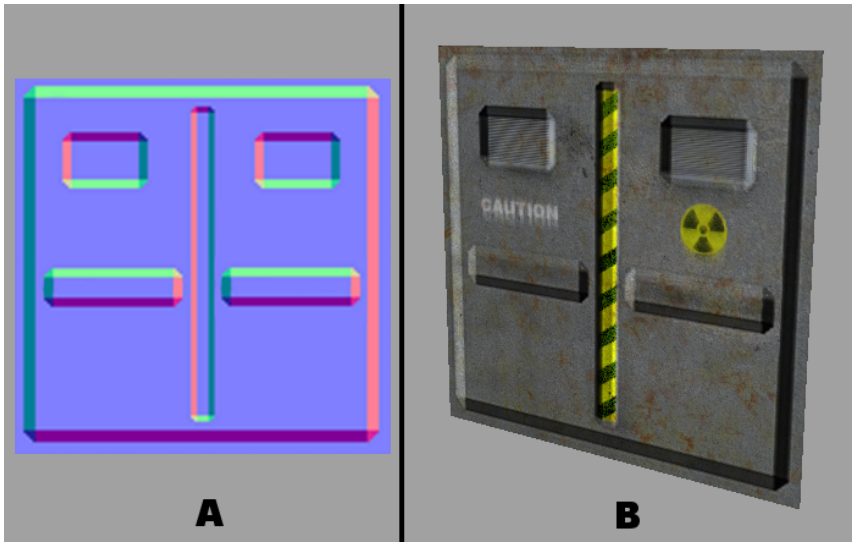

Figura 42: Geração da textura pelo novo sistema de pintura: A) *Normal map* criado. B) Texturas aplicadas ao plano poligonal. A Textura do canal de cor utilizada é a mesma usada na Figura 41. (Tempo total estimado do procedimento: 2 minutos e 20 segundos.)

coordenadas de mapeamento permitem que certas partes da textura sejam repetidas em determinadas faces do objeto, recurso muito utilizado em modelos 3D usados em jogos digitais, com objeto de otimizar a resolução da textura. A Figura 44 ilustra a recriação do relevo utilizando o protótipo.

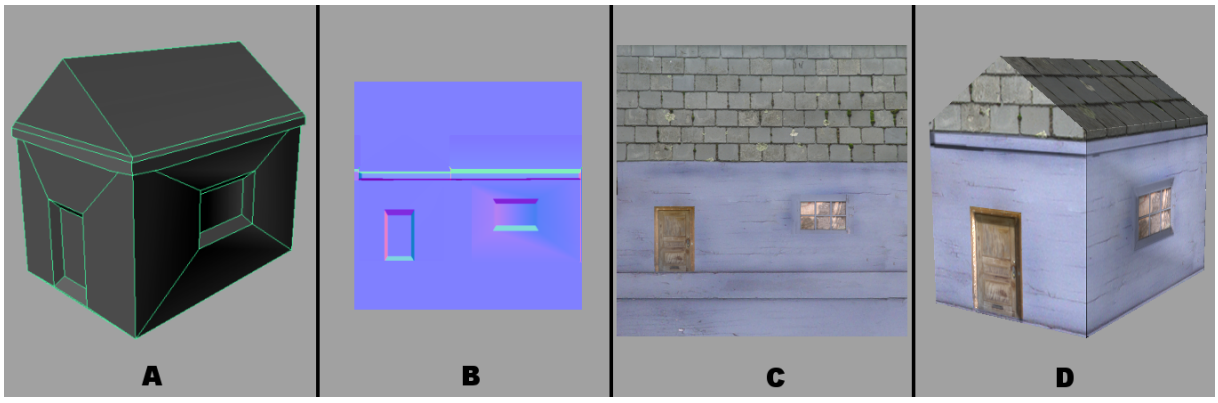

Figura 43: Geração da textura pelo processo tradicional: A) Modelagem de referência. B) *Normal map* criado. C) Textura desenvolvida em um editor de imagens, para ser aplicada ao canal de cor no objeto 3D. D) Texturas aplicadas ao objeto poligonal. (Tempo total estimado do procedimento: 3 minutos e 40 segundos. Este tempo não considera o desenvolvimento da textura extra aplicada ao canal de cor.)

A Figura 45 demonstra um objeto que representa um painel de controle, com alguns elementos em forma circular. Para a modelagem de referência no método tradicional de criação do *normal map*, foi utilizado cilindros modificados, sobre um plano poligonal com faces subdivididas. Já pelo novo sistema de pintura, foi utilizada a ferramenta que permite "pincelar" círculos e retângulos. A Figura 46 mostra o resultado.

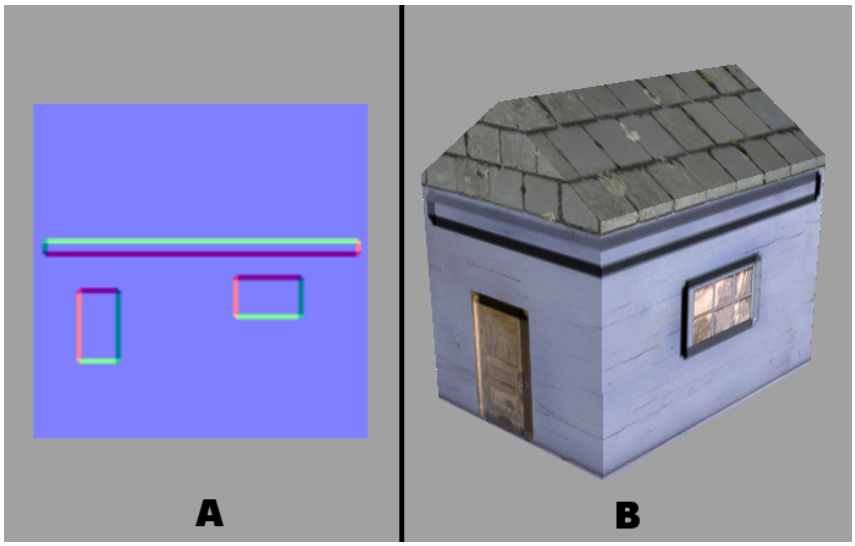

Figura 44: Geração da textura pelo novo sistema de pintura: A) *Normal map* criado. B) Texturas aplicadas ao objeto poligonal. A Textura do canal de cor utilizada é a mesma usada na Figura 43. (Tempo total estimado do procedimento: 2 minutos.)

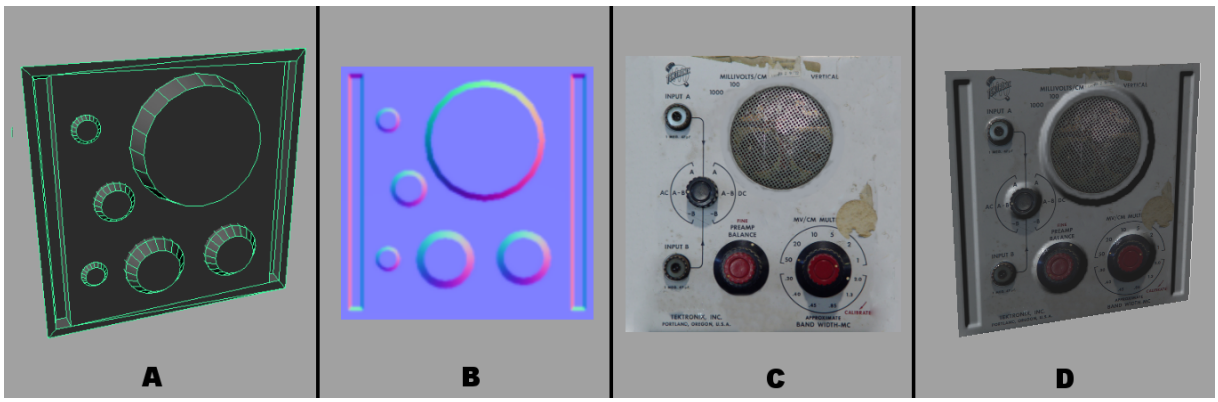

Figura 45: Geração da textura pelo processo tradicional: A) Modelagem de referência. B) *Normal map* criado. C) Textura desenvolvida em um editor de imagens, para ser aplicada ao canal de cor no objeto 3D. D) Texturas aplicadas ao objeto poligonal. (Tempo total estimado do procedimento: 8 minutos. Este tempo não considera o desenvolvimento da textura extra aplicada ao canal de cor.)

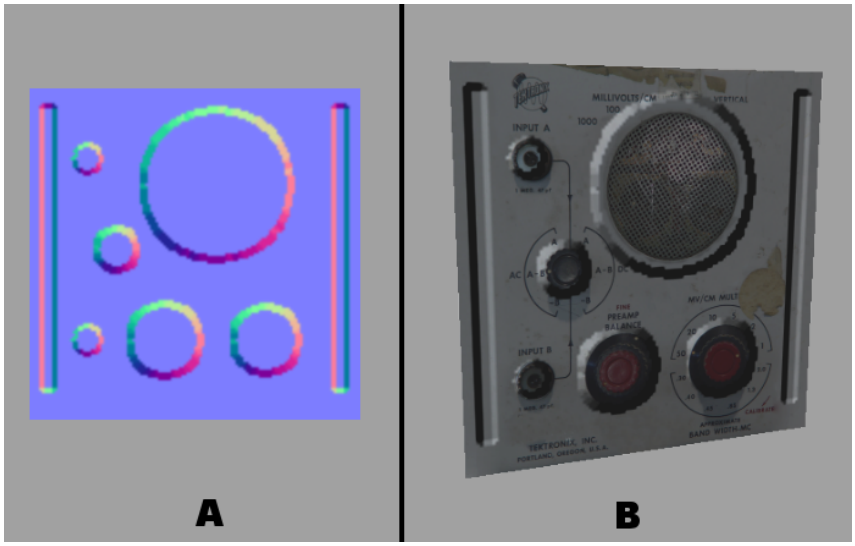

Figura 46: Geração da textura pelo novo sistema de pintura: A) *Normal map* criado. B) Texturas aplicadas ao objeto poligonal. A Textura do canal de cor utilizada é a mesma usada na Figura 45. (Tempo total estimado do procedimento: 5 minutos.)

A Figura 47 exibe um tanque de guerra, modelado com poucos polígonos. Este objeto teve seu *normal map* criado somente com o sistema de pintura, sem utilizar a modelagem de referência do método tradicional. O tempo de criação não foi computado, pois trata-se de um objeto mais complexo, desenvolvido com ênfase na qualidade de modelagem e textura, sem priorizar o tempo gasto nestas tarefas.

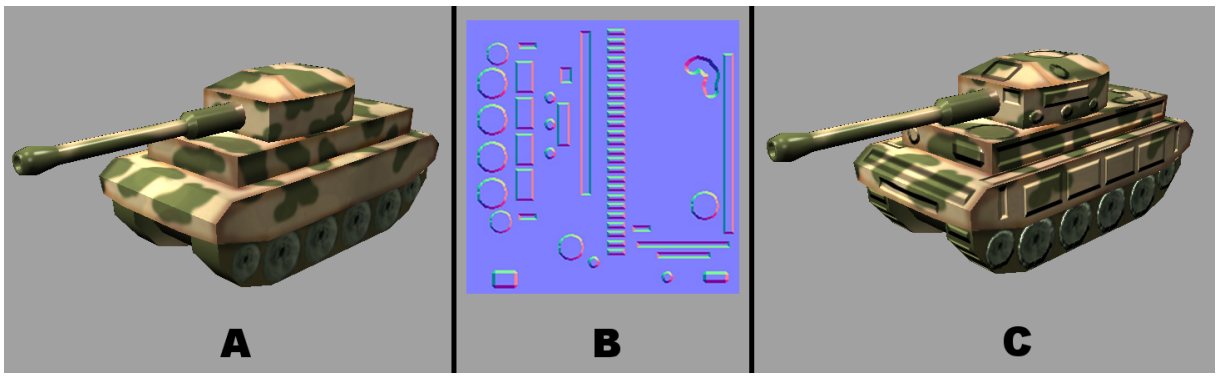

Figura 47: Tanque de guerra: A) Modelo simplificado com somente textura de cor. B) *Normal map* criado pelo protótipo. C) Modelo com *normal map* aplicado.

Sobre a animação de *normal maps*, como exemplo foi animado um relevo simples, simulando um botão sendo pressionado. Esta animação ocorreu no espaço de quatro frames. As texturas foram aplicadas em um plano poligonal, sendo que além do *normal map* criado e animado pelo protótipo, foi elaborada uma textura extra para ser usada no canal de cor, para ilustração (Figura 48). Para cada frame um *normal map* diferente foi criado. A Figura 49 demonstra as texturas criadas pela interpolação linear. O resultado final das texturas aplicadas no plano pode ser observado na Figura 50.

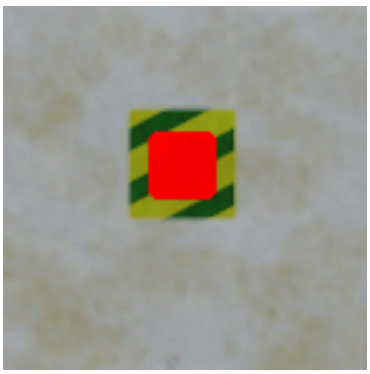

Figura 48: Textura desenvolvida em um editor de imagens para ser aplicada ao canal de cor no plano poligonal.

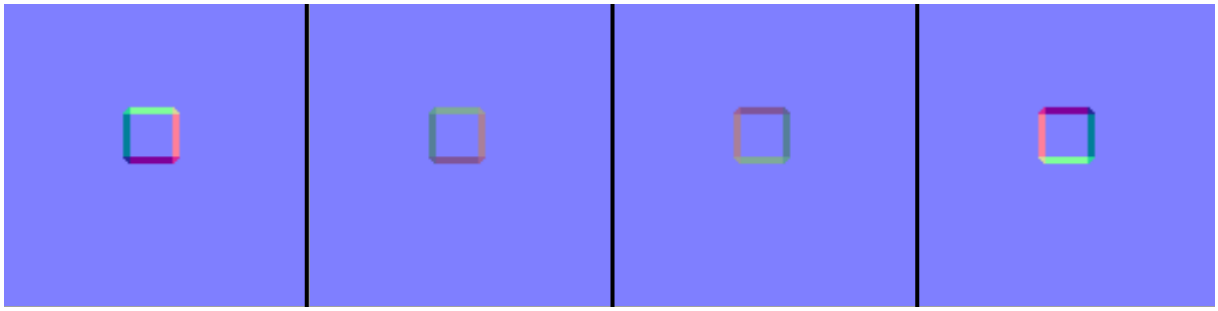

Figura 49: *Normal maps* criados por interpolação linear no protótipo. Escrita uma textura por frame.

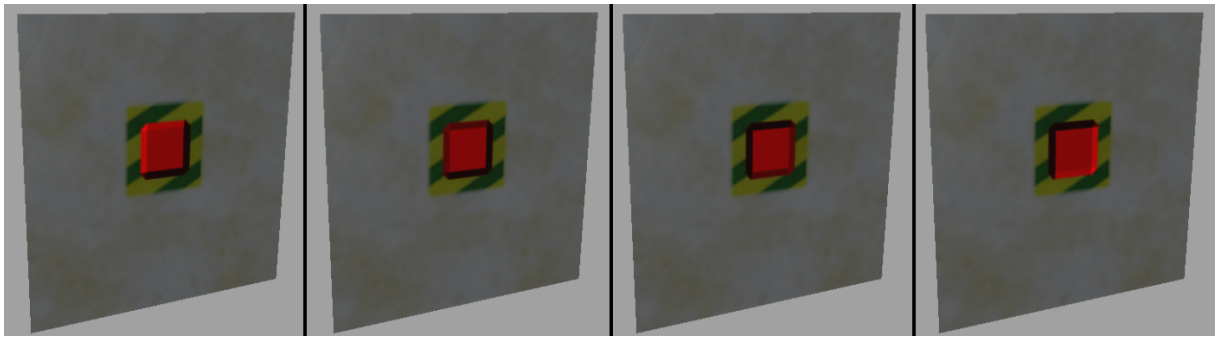

Figura 50: Texturas aplicadas ao plano poligonal. Em cada frame da animação um *normal map* respectivo é criado. Ao executar a seqüência em determinada velocidade, nota-se a animação do relevo.

5.1 Limitações

Uma das limitações do protótipo consiste na forma de geração *pixel* a *pixel* do *normal map*, pelo método desenvolvido no *plug-in* que roda dentro do Maya. Em certos casos, principalmente em desenhos mais circulares ou arqueados, o *normal map* gerado pode apresentar bordas serrilhadas de suas formas, como pode ser observado na Figura 46, exibida anteriormente neste capítulo.

O sistema de geração tradicional da textura, que compara o modelo 3D simplificado com o de referência, realizado pela ferramenta *"surface sampler"* do Maya, consegue gerar o *normal map* com bordas mais suaves. Este fato se deve a escrita do arquivo usar precisão de $sub - pixel$ na imagem final, ao contrário do protótipo que desenha cada *pixel* de forma independente. A reescrita do *plug-in* levando isto em consideração deve resolver este problema.

O protótipo em seu estágio atual de desenvolvimento busca demonstrar o novo conceito de pintura de normal maps, mas melhorias nestes aspectos de escrita da textura devem ser realizadas para se atingir a mesma qualidade existente no método tradicional de geração destas imagens.

Outra limitação existente no protótipo é a interface. Não há muita precisão em onde pintar as formas no normal map, assim como seria interessante a possibilidade de duplicar estas formas em outras partas da textura, configurar os parâmetros das ferramentas através de um método mais gráfico e intuitivo com uma pré-visualização do efeito, maior agilidade em alterar qualquer forma já pintada, entre outras melhorias. O tempo estimado de trabalho com o protótipo acaba sendo prejudicado por causa da interface atual. Também para gerar animações, a interface não permite muitas opções de alterações de formas já pintadas, restringindo as possibilidades na criação dos *keyframes*.

Além disto as ferramentas que foram desenvolvidas não permitem gerar *normal maps* que representam relevos mais orgânicos. Formas que não possuam bordas bem definidas não são realizáveis pelo protótipo neste estágio de desenvolvimento. Outras ferramentas devem ser agregadas para este fim, expandindo as possibilidades de desenho no *normal map*, de forma que consiga simular os efeitos de uma maior gama de geometrias de referência diversas.

Um outro método de geração de *normal maps*, que tem características semelhantes com este trabalho por não depender do sistema tradicional de modelagem e lidar diretamente com processamentos 2D, foi descrito no capítulo 2, seção 2.2. Trata-se do trabalho "Colorização 3D para Animação 2D" (BEZERRA, 2005), que descreve um sistema de geração automática de *normal maps* para desenho 2D. A próxima seção comenta a implementação realizada sobre este método.

5.2 Geração de *normal maps* em desenho 2D

Além dos exemplos apresentados, foi implementado, de forma não integrada ao protótipo discutido, um sistema de geração automática de *normal maps* para desenho 2D.

A geração de um *normal map* estimado pela forma do desenho acontece primeiramente pela extração das curvas. Considera-se um desenho que possua linha com somente um *pixel* da largura, para simplificação. Usa-se o método *chain code* (PARKER, 1994), que é uma forma simples e eficiente de obter a informação sobre a direção da linha do desenho. Funciona da seguinte forma: armazena-se em um vetor a posição X e Y de um *pixel* que forma a linha; após, registra-se um número que representa a direção do próximo *pixel* adjacente, em sentido horário. Utiliza-se somente 8 direções possíveis, numeradas em ordem crescente também em sentido horário. Ou seja, o número 1 representa a direção superior, o 2 representa a diagonal superior direita, o 3 a direção direita e assim por diante. Desta forma, tem-se um vetor que descreve toda a direção da linha do desenho.

Usando as informações deste vetor, o próximo passo é determinar a normal de cada *pixel* da linha do desenho, estimados pela direção da mesma. Toma-se o vetor perpendicular ao vetor direção da curva, que aponta para fora do interior da curva, permitindo que depois de interpoladas as normais, estes definam uma superfície convexa. A escolha da direção da normal é importante, pois se for usada a normal perpendicular que aponta para o interior da curva, teremos depois da interpolação a definição de uma superfície côncava, que não é desejável para gerar volume ao desenho.

Os vetores normais podem estar em qualquer espaço, porém é conveniente transformá-los para o espaço da tela (X,Y) com profundidade Z. Para uma projeção ortográfica, a componente Z das normais ao longo da aresta de uma superfície curva deve ser zero para manter a normal perpendicular ao vetor do olho. A Figura 51 demonstra a forma de identificação dos vetores normais a uma curva.

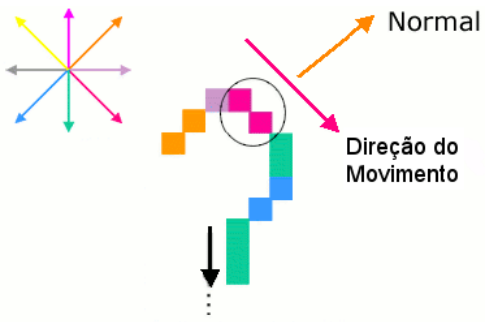

Figura 51: Identificação dos vetores normais a uma curva. (BEZERRA, 2005)

Foram usadas somente 8 posições possíveis de direção para preencher o *chain code*, por simplificação. Isto gerou normais que podem não oferecer uma transição adequada em relação às outras normais adjacentes, especialmente em partes curvadas da linha no desenho. Desta forma, há necessidade de normais com direções intermediárias, para gerar um campo de normais mais suave, após a interpolação. Por isso, deve-se realizar um procedimento de suavização: para cada ponto, o vetor normal é calculado pela adição dos vetores normais de seus pontos vizinhos anteriores e posteriores. Isto ajuda a eliminar transições abruptas nas normais que formam a linha.

O passo final para a geração do *normal map* a partir do desenho, é a interpolação das normais que formam a linha, preenchendo o interior do desenho com um campo de normais que representa o volume aproximado. Como dito anteriormente, este volume será convexo, como se a forma definida no desenho tenha relevo esférico. O centro geométrico

do desenho terá a normal apontando em direção ao observador, que será interpolada com as normais que já foram estimadas, criando transições entre elas.

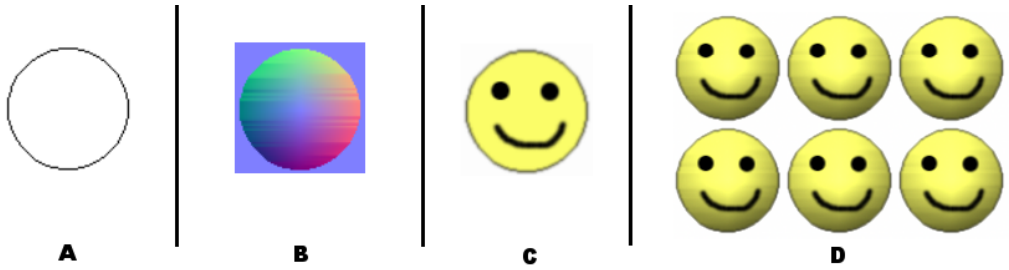

Figura 52: A) Desenho original. B) Mapa de normais gerado automaticamente. C) Textura de cor. D) Renderização feita no Maya, aplicando textura de cor e mapa de normais em um plano poligonal (repetido 6x).

Após o campo de vetores normais ser criado, a área externa do desenho até a borda da imagem é preenchida com normais "frontais", que apontam para o observador. Esta área não representa nenhuma alteração nas normais, serve apenas para completar o *normal map*.

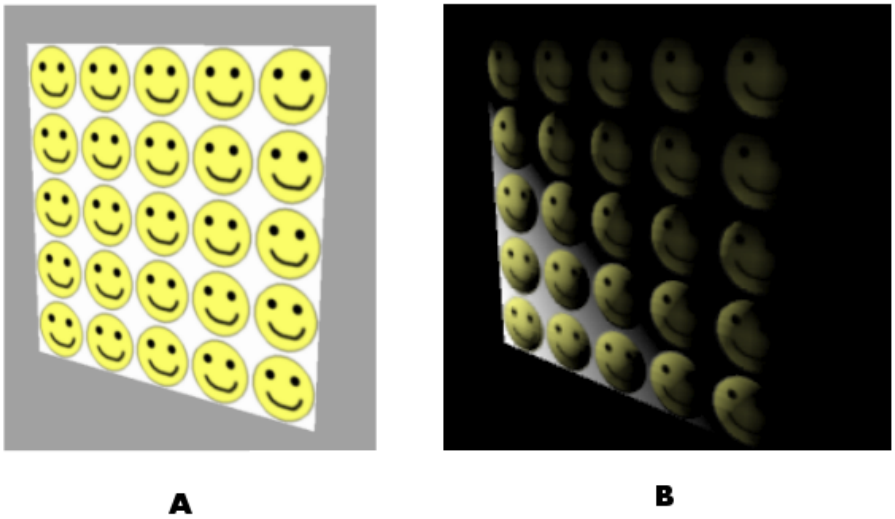

Figura 53: A) Plano poligonal inclinado só com textura de cor. B) Mesmo plano com *normal map* aplicado e iluminado de forma a ressaltar o volume.

As Figuras 52,53 e 54 demonstram exemplos de desenhos, seus *normal maps* gerados e renderizações de teste feitos no *software* Maya. Na renderização da Figura 53 foi utilizado luzes na cena 3D para realçar o efeito de volume do *normal map*. Além disso, em todos os testes, outra textura de cor foi aplicada no plano, que seriam os detalhes que o artista quer representar no desenho final. Assim o desenho inicial das linhas serve para estimar as normais, e a textura de cor serve para detalhar a renderização.

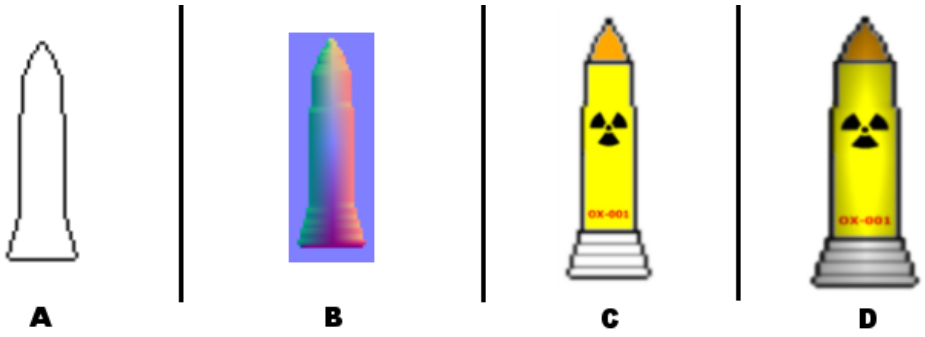

Figura 54: A) Desenho original. B) Mapa de normais gerado automaticamente. C) Textura de cor. D) Renderização feita no Maya, aplicando textura de cor e mapa de normais em um plano poligonal.

6 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este trabalho apresenta um novo modelo para criar *normal maps*, utilizando ferramentas que se assemelham a um sistema de pintura. Estas alterações são efetuadas diretamente no modelo poligonal, onde o usuário observará o efeito em tempo computacional interativo.

Este protótipo visa auxiliar o complexo processo de criação de *normal maps*, onde o método tradicional requer a modelagem de um objeto 3D extra, usado como referência geométrica para criação da textura. Como demonstrado pelos resultados, em certas situações o conceito de pintar diretamente sobre o objeto simplificado, criando o relevo do mesmo, consegue tempos de produção diferenciados em relação ao processo original. Porém para simulação de modelagens mais complexas ou orgânicas, o uso tradicional de um segundo modelo de referência ainda é necessário.

Assim, uma solução híbrida poderá ser eficiente, criando geometria onde for requerido, e agilizando o restante do processo com o novo sistema de pintura. Pois a solução de recriar qualquer relevo baseado em ferramentas de pintura é complexa, sendo que o processo tradicional de criação de *normal maps*, apesar de requerer considerável tempo do usuário, permite controle total sobre a textura gerada a partir desta geometria.

Este trabalho pode ser evoluído pela implementação de novas ferramentas de pintura, para satisfazer uma maior gama de tipos de relevo possíveis de serem criados, com facilidade. Formas geométricas 2D, como trapézios, losangos, entre outras, além de polígonos de forma livre, poderiam ser úteis para criar relevo baseado nestas formas.

Outros facilitadores para desenhar um relevo mais complexo seriam a possibilidade de gerar intersecções e subtrações de pinceladas, além da reutilização de formas já adicionadas em outras partes da textura. Em muitos casos um certo tipo de relevo costuma aparecer em várias áreas, somente com escala diferenciada.

A interface poderia ser mais eficiente e intuitiva, com retorno visual mais sofisticado em relação a configuração da ação que o usuário executa naquele momento. Desta forma, poderia haver um ajuste da ferramenta utilizada, bastando clicar em controles sobre o ícone dela, ao invés da necessidade de entrar valores numéricos em campos da interface distantes do objeto sendo "pintado".

Além disto, a velocidade do protótipo poderia ser amplamente otimizada com uma melhor comunicação entre o script escrito em MEL e o *plug-in* que cria a textura em disco. Otimizar cálculos também agilizaria todo o processamento, pois a performance não

entrou no escopo deste trabalho, havendo grande margem para melhorias nesta área.

REFERÊNCIAS

AGRAWALA, M.; BEERS, A. C.; LEVOY, M. 3d painting on scanned surfaces. In: *SI3D '95: Proceedings of the 1995 symposium on Interactive 3D graphics*. New York, NY, USA: ACM Press, 1995. p. 145–ff.

BEZERRA, H. M. *Colorização 3D para Animação 2D*. Dissertação (Mestrado) — Pontifícia Universidade Católica do Rio de Janeiro, 05 2005.

BLINN, J. F. Simulation of wrinkled surfaces. In: *SIGGRAPH*. [S.l.]: ACM Press, 1978. p. 286–292.

COOK, R. L. Shade trees. In: *SIGGRAPH '84: Proceedings of the 11th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press, 1984. p. 223–231.

DAILY, J.; KISS, K. 3d painting: paradigms for painting in a new dimension. In: *CHI '95: Conference companion on Human factors in computing systems*. New York, NY, USA: ACM Press, 1995. p. 296–297. ISBN 0-89791-755-3.

ENTERTAINMENT-SOFTWARE-ASSOCIATION. Essential facts about the computer and video game industry. 2006. http://www.theesa.com.

FOURNIER, A. Normal distribution functions and multiple surfaces. In: *Graphics Interface '92 Workshop on Local Illumination*. Vancouver, BC, Canada: [s.n.], 1992. p. 45–52.

GREGORY, A. D.; EHMANN, S. A.; LIN, M. C. intouch: Interactive multiresolution modeling and 3d painting with a haptic interface. In: *VR '00: Proceedings of the IEEE Virtual Reality 2000 Conference*. Washington, DC, USA: IEEE Computer Society, 2000. p. 45. ISBN 0-7695-0478-7.

HARO, B. G. A.; ESSA, I. (Ed.). *Real-time, Photo-realistic, Physically Based Rendering of Fine Scale Human Skin Structure*. [S.l.]: 12th Eurographics Workshop on Rendering, London, England, 2001.

HORN, B.; BROOKS, M. *Shape from Shading*. [S.l.]: MIT Press, 1989.

JAIN RANGACHAR KASTURI, B. G. S. R. *Machine Vision*. [S.l.]: Mc Graw Hill, 1995.

KILGARD, M. J. A practical and robust bump-mapping technique for today´s gpus. *Game developers conference*, Advanced OpenGL game development, 2000.

MCGUIRE, M.; FEIN, A. Real-time cartoon rendering of smoke and clouds. In: *NPAR*. [S.l.: s.n.], 2006.

OLIVEIRA, M. M.; BISHOP, G.; MCALLISTER, D. Relief texture mapping. In: *SIGGRAPH 2000: Proceedings of the 27th annual conference on Computer graphics and interactive techniques*. New York, NY, USA: ACM Press/Addison-Wesley Publishing Co., 2000. p. 359–368.

PARKER, J. R. *Practical Computer Vision using C*. [S.l.]: John Wiley & Sons, New York, 1994.

POLICARPO, F.; OLIVEIRA, M. M.; COMBA, J. L. D. Real-time relief mapping on arbitrary polygonal surfaces. In: *SI3D 2005: Proceedings of the 2005 symposium on Interactive 3D graphics and games*. New York, NY, USA: ACM Press, 2005. p. 155–162.

TASDIZEN, T. et al. Geometric surface processing via normal maps. *ACM Trans. Graph.*, v. 22, n. 4, p. 1012–1033, 2003.