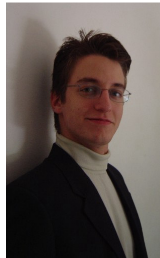

F809 – Relatório Final de Progresso

Representação de Fenômenos em três dimensões:

Interferência de Ondas Bidimensionais

Por David D. Chinellato
Orientador: Engo. Pedro Raggio

F809 – Relatório Final de Progresso

Representação de Fenômenos em três dimensões

Interferência de Ondas Bidimensionais

1. Introdução

Neste relatório, será exposto todo o princípio de funcionamento do programa desenvolvido para a exibição das ondas bidimensionais com efeitos de perspectiva e sombreamento. A finalidade deste documento é evidenciar o funcionamento básico por trás do programa que será adaptado ao ensino de sistemas nos quais ocorram interferências entre duas ondas bidimensionais; um exemplo bastante interessante, o de um sistema de dupla fenda, será mostrado.

Inicialmente, será explicado o procedimento de projeção bidimensional de um ponto qualquer do espaço tridimensional. Este procedimento de projeção bidimensional será então base para o desenvolvimento de uma rotina de sombreamento adequada à percepção de tridimensionalidade que pretendemos exibir em uma imagem no monitor de um computador ou impressa. Finalmente, as ondas bidimensionais que queremos representar devem ser compostas de uma quantidade grande de triângulos de maneira tal que haja percepção de uma superfície contínua e suave; o sombreamento e projeção devem, portanto, ser aplicados a uma forma paramétrica de onda bidimensional.

2. O algoritmo de projeção 3D→2D

Uma parte bastante interessante desta empreitada é o desenvolvimento de um modo para representar um ponto no espaço tridimensional dentro de um espaço mais limitado, de apenas duas dimensões.

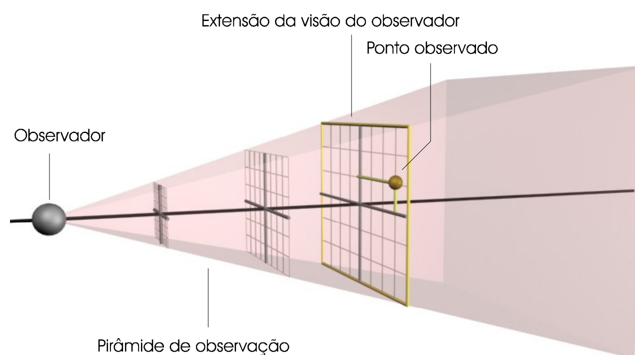

Figura 1: O mapeamento de uma pirâmide de pontos observáveis na tela.

Portanto, matematicamente, a primeira coisa que se deve obter é a criação de uma metodologia para representar uma projeção adequada de um dado ponto no espaço tridimensional (x,y,z) em um ponto em uma tela de computador *(u,v)*[1], com *u* sendo posição em relação a um eixo horizontal e *v* sendo posição em relação a um eixo vertical. As figuras 1 e 2 auxiliam na compreensão do que será feito. Desta forma, cada ponto deve ter apenas dois graus de liberdade no espaço projetado, o que faz sentido. Para cada ponto deve ser atribuído também um valor de profundidade, o que adicionará uma coordenada extra que não pode ser representada, mas que será utilizada na projeção de maneira a ser ainda pesquisada. Desta forma, deveremos criar um procedimento que tem como entrada as coordenadas de um ponto no espaço tridimensional *(x,y,z)* e a posição e características do observador (a "câmera") e retorna as coordenadas de um ponto *(u,v,w)* no espaço bidimensional, com *w* sendo um valor real e positivo referente à profundidade do ponto em relação à câmera. Esta

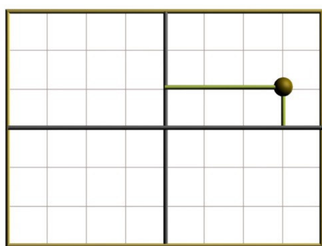

Figura 2: O ponto da figura 1 segundo o observador.

[1] Note que *u* e *v* têm de ser números inteiros sujeitos às limitações de resolução do monitor e do computador utilizado. Tipicamente, um sistema de alta resolução admitiria $0<u<1024$ e $0<v<768$ para uma resolução vertical x horizontal de 1024x768 pixels, um valor tradicional de boa resolução.

última coordenada é apenas auxiliar neste ponto, e pode vir a ser útil para atribuir características de profundidade.

3. O algoritmo de projeção 3D→2D: A Matemática Envolvida

Desenvolveremos aqui a transformação descrita qualitativamente na seção anterior matematicamente. A transformação certamente pode ser escrita como:

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = F\begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3.1)$$

Na expressão (2.1), a função F é um campo vetorial que relaciona as três coordenadas x,y,z com u,v,w. Se tivéssemos como objetivo uma projeção isométrica do espaço, isto é, sem efeito de perspectiva – correspondente a um observador no infinito – a função F corresponderia a uma simples mudança de base; veremos que uma projeção com perspectiva não é muito diferente disto, exceto por uma pequena modificação.

A mudança de base x,y,z →u,v,w que é equivalente à projeção isométrica do espaço tridimensional no espaço bidimensional corresponde a uma mudança de base considerando os vetores unitários padrão na direção x,y e z e os vetores unitários definidos em relação à direção de observação da forma:

$$\begin{cases} \hat{x} \\ \hat{y} \\ \hat{z} \end{cases} \rightarrow \begin{cases} \hat{u} - direção \quad horizontal \quad no \quad monitor \\ \hat{v} - direção \quad vertical \quad no \quad monitor \\ w - profundidade \end{cases} \quad (3.2)$$

Esta mudança está evidenciada na figura 3, onde podem ser vistos os vetores unitários na direção u e v (em cinza) e os vetores unitários padrão x,y e z. Note que o paralelepípedo transparente demarca a região observável pelo monitor, a região que poderá ser desenhada. Esta projeção é dita isométrica, pois uma determinada distância no espaço

projetado (u,v,w) tem a mesma relação com a distância em (x,y,z), independente da posição. Desta forma, para a projeção isométrica, basta pensar em F da relação (3.1) como sendo uma matriz de mudança de base na qual as colunas da matriz correspondem aos vetores $\hat{x}, \hat{y}, \hat{z}$ escritos na base nova, $\{\hat{u}, \hat{v}, \hat{w}\}$. Esta mudança – a descrição de um ponto nas coordenadas u,v e w no lugar de x,y e z - está representada na expressão (3.3) e está de acordo com álgebra linear básica. Consideramos os vetores como sendo unitários, o que conserva o módulo em uma transformação do tipo (3.3).

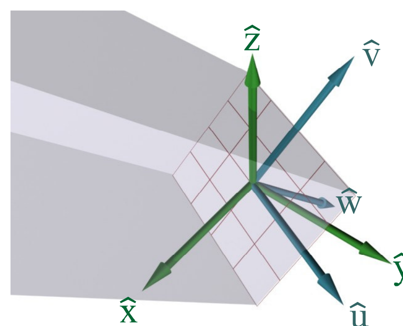

Figura 3: A Mudança de base para perspectiva isométrica.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} \hat{x}_u & \hat{y}_u & \hat{z}_x \\ \hat{x}_v & \hat{y}_v & \hat{z}_y \\ \hat{x}_w & \hat{y}_w & \hat{z}_z \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3.3)$$

Já que cada componente da matriz é uma projeção específica, podemos reescrever (3.3) como (3.4), onde temos produtos escalares entre os vetores ortonormais que definem as duas bases. Esta forma é a mais completa para a mudança de base que usaremos.

$$\begin{pmatrix} u \\ v \\ w \end{pmatrix} = \begin{pmatrix} (\hat{x}\cdot\hat{u}) & (\hat{y}\cdot\hat{u}) & (\hat{z}\cdot\hat{u}) \\ (\hat{x}\cdot\hat{v}) & (\hat{y}\cdot\hat{v}) & (\hat{z}\cdot\hat{v}) \\ (\hat{x}\cdot\hat{w}) & (\hat{y}\cdot\hat{w}) & (\hat{z}\cdot\hat{w}) \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} \quad (3.4)$$

Vê-se imediatamente na expressão (3.4) que as linhas da matriz de mudança de base também denotam os vetores da base nova escritos nas coordenadas da base original, x,y e z.

A implementação desta operação seria simples; porém, é necessário ainda considerar um fator adicional para se obter um efeito mais realista de perspectiva por parte de um observador real situado a uma distância finita do objeto observado. Desta forma, será realizada uma operação de perspectiva real que tem fundamentos em ótica básica e é

matematicamente bastante simples: vamos apenas dividir as coordenadas u e v pela coordenada w deslocada: vamos posicionar os vetores unitários da base $\{\hat{u}, \hat{v}, \hat{w}\}$ para que estes estejam na mesma posição do observador. Desta forma, a coordenada w denota a distância do observador ao ponto observado quando medida sobre o eixo de observação da pirâmide visível, como indica a figura 1. A figura 1 também indica uma grade de 8x6 quadrados (trabalharemos com uma proporção de figura bidimensional de saída de 4:3, pois esta é a proporção amplamente utilizada para monitores) que depende linearmente com a distância devido à ótica do problema; basta considerar regras simples de proporcionalidades de triângulos, como na figura 4, e vê-se logo que a proposta de perspectiva real exposta na expressão (3.5) é razoável. Note que esta não é mais uma operação linear de mudança de base, mas sim uma operação diferente, que terá como resultado coordenadas u,v e w com perspectiva de um observador a uma distância finita do ponto (x,y,z) observado.

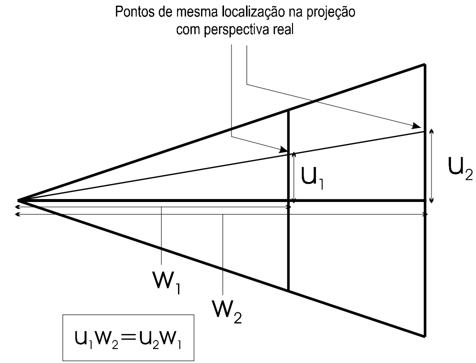

Figura 4: Corte da Pirâmide visível na figura 1.

$$\begin{cases} (não-lineares)\begin{cases} u(x,y,z)=\left(\dfrac{x\hat{x}_u + y\hat{y}_u + z\hat{z}_u}{x\hat{x}_w + y\hat{y}_w + z\hat{z}_w}\right) \\ v(x,y,z)=\left(\dfrac{x\hat{x}_v + y\hat{y}_v + z\hat{z}_v}{x\hat{x}_w + y\hat{y}_w + z\hat{z}_w}\right) \end{cases} \\ w(x,y,z)= x\hat{x}_w + y\hat{y}_w + z\hat{z}_w \end{cases} \qquad (3.5)$$

As operações necessárias para obter as novas coordenadas u e v não são lineares como a para obtenção de w.

O algoritmo implementado para a projeção de pontos utiliza exatamente este método para calcular as novas coordenadas. O procedimento escrito em Free Pascal pode ser visto no Apêndice A deste relatório com o nome de *Morph* (do inglês para mudança de forma). Este procedimento calcula a perspectiva de maneira ainda isométrica, e o procedimento *Dimkill* (de *dimension kill*) faz a correção para perspectiva realista.

Com este procedimento, podemos converter qualquer ponto no espaço tridimensional em um ponto em uma projeção bidimensional com perspectiva, semelhante àquilo que um observador veria.

4. O sombreamento

Abordaremos agora um problema diferente: como representar uma superfície no espaço tridimensional de maneira suficientemente informativa na nossa projeção bidimensional? Uma idéia bastante útil é dividir a superfície em um grande número de triângulos; porém, isto levanta a questão de que coloração deve se atribuir a cada um dos triângulos. Para esclarecer o assunto, concebemos uma metodologia que atribui uma cor a cada triângulo através da quantidade de luz incidente em cada um destes triângulos a partir de uma certa fonte de luz de posição tridimensional definida.

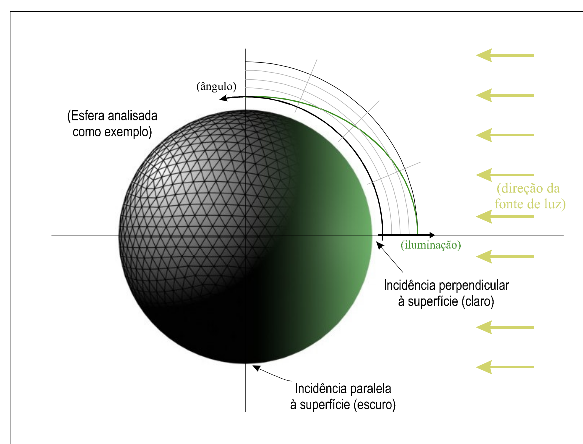

Figura 5: O efeito de sombreamento por quantidade de luz incidente por unidade de área.

Para quantizar a luz que o observador recebe de cada um dos triângulos, pode-se pensar no co-seno do ângulo entre a incidência de luz e a normal da superfície. Isto dá uma medida da quantidade de luz incidente, em relação à máxima possível (incidência perpendicular à superfície). No nosso caso, fizemos uso do sistema de cores RGB da forma (r,g,b), com r = g = b inteiros de 0 a 255, ou seja, consideramos apenas 256 tonalidades de cinza. A essência do raciocínio utilizado na figura 5 foi sintetizada em uma expressão que é função do ângulo entre a fonte de luz, o triângulo e o vetor normal à superfície que retorna um valor inteiro de 0 a 255 que será colocado nas coordenadas de cor (r,g,b). Para obter o ângulo necessário para a avaliação do valor desta expressão para cada um dos triângulos, consideramos simplesmente o produto escalar entre o vetor normal de cada triângulo e o vetor definido pela posição do triângulo subtraída da

posição da fonte de luz. Se todos os vetores considerados estiverem devidamente normalizados, o resultado deste produto escalar será automaticamente o co-seno do ângulo relevante ao cálculo de iluminação; desta forma, é até mais conveniente redefinir a função de sombreamento para ter como entrada não o ângulo, mas o co-seno do ângulo entre o vetor de incidência de luz e o vetor normal.

Algebricamente, se considerarmos os vértices do triângulo como sendo denotados por (x_i, y_i, z_i), com i = 1, 2 e 3 para cada um dos três vértices, o vetor normal $\vec{v}_n$ pode ser escrito como:

$$\vec{v}_n = \pm \begin{pmatrix} x_1 - x_3 \\ y_1 - y_3 \\ y_1 - y_3 \end{pmatrix} \times \begin{pmatrix} x_2 - x_3 \\ y_2 - y_3 \\ y_2 - y_3 \end{pmatrix} \quad (4.1)$$

O sinal de $\vec{v}_n$ é de grande relevância e deve ser determinado de superfície em superfície analisada. Resolveremos isto na parte paramétrica da definição de superfície.

Para encontrar o vetor de incidência de luz $\vec{v}_{incidência}$, basta avaliar a expressão dada por (4.2), onde (x_L, y_L, z_L) denota a posição de uma fonte de luz definida pelo usuário antes da execução do programa.

$$\vec{v}_{incidência} = \frac{1}{3} \begin{pmatrix} x_1 + x_2 + x_3 \\ y_1 + y_2 + y_3 \\ z_1 + z_2 + z_3 \end{pmatrix} - \begin{pmatrix} x_L \\ y_L \\ z_L \end{pmatrix} \quad (4.2)$$

Note que estes vetores devem ser apropriadamente normalizados; este é um procedimento simples, mas deve-se atentar a não colocar uma fonte de luz muito próxima a qualquer triângulo ou considerar triângulos pequenos demais, pois isto pode fazer com que a tentativa de normalizar estes vetores resulte em uma divisão por zero (isto ocorre no caso de a máquina não ter a precisão ao distinguir o valor de um módulo muito próximo de zero de um valor realmente nulo). Na implementação que foi concebida para este projeto, o procedimento *Illumina* é responsável pelo cálculo do co-seno do ângulo entre a normal e a

incidência de luz de cada triângulo; uma função apropriadamente escolhida *L(Illumina)* atribui então uma cor a cada triângulo baseado nesta informação.

Apenas os fatores mostrados aqui já permitem a criação de um programa que projeta e sombreia adequadamente qualquer superfície. O programa desenvolvido até este ponto do projeto utiliza estes princípios básicos aliados ainda a alguns procedimentos bastante simples para determinar os parâmetros de entrada como triângulos de uma superfície. Repassaremos no apêndice A deste relatório exatamente o que cada um destes passos faz; note que os passos mais complexos são, de fato, os procedimentos *Illumina, L, Morph* e *Dimkill*. Os outros procedimentos podem perfeitamente ser compreendidos com a adição de poucos comentários, como pode ser visto no apêndice A.

5. Estado do programa

Resumidamente, o programa desenvolvido até aqui é capaz de representar uma superfície paramétrica na qual se encontram duas funções genéricas de onda em um monitor de maneira a simular tridimensionalidade desta mesma superfície, como indicado na figura 6. O programa faz uso extensivo dos procedimentos cujo funcionamento está explicado nas seções anteriores e cujos detalhes podem ser lidos no Apêndice A; note que a superfície é composta por cerca de 130 mil triângulos, cada um iluminado individualmente.

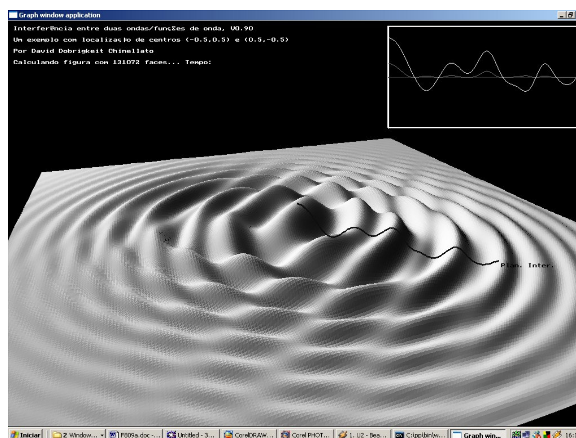

Figura 6: *Screenshot* do programa em ação.

O programa já estava operacional quando foi feito o relatório parcial; porém, foi adicionada uma funcionalidade extra. Agora, o programa dispõe de um pequeno menu de opções em texto que permite a manipulação de todos os parâmetros envolvidos na criação das ondas. Os parâmetros variáveis pelo usuário são lidos de um arquivo params.txt e é então perfeitamente possível

ao usuário modificar qualquer parâmetro do programa. A manipulação adequada destes parâmetros é de bastante importância para a didática do problema, já que um eventual aluno seria capaz de modificar o programa e adaptá-lo aos seus desejos. Para a compreensão destes parâmetros, é necessário entender a função de onda considerada internamente pelo programa, que é da forma 5.1.

$$f(x,y) = Ae^{-B\sqrt{(x-xcen)^2+(y-ycen)^2}} sin\left(w\sqrt{(x-xcen)^2+(y-ycen)^2} + \phi + t \right) \quad (5.1)$$

Uma versão diferente do programa, disponível também, faz uso da função um pouco diferente dada por 5.2. Esta função é mais adequada para a demonstração intuitiva e qualitativa, pois produz imagens bastante chamativas e informativas, se bem que não tão precisas, enquanto 5.1 é a mais adequada para casos mais convencionais.

$$f(x,y) = Ae^{-B\left((x-xcen)^2+(y-ycen)^2\right)} sin\left[w\left((x-xcen)^2+(y-ycen)^2\right) + \phi + t \right] \quad (5.2)$$

Para o caso em que não há atenuação presente, basta colocar A=0. Para o caso em que se deseja mostrar apenas uma gaussiana em duas dimensões, pode-se usar 5.2 com B escolhido convenientemente e w escolhido como 0, com ângulo de fase igual a cerca de 1,5707 (metade de π). Isto deve tornar a utilidade do programa bastante evidente. Os parâmetros variáveis incluem, na ordem em que aparecem no arquivo de parâmetros params.txt, as seguintes grandezas:

- *<t0>*
    - *Tempo inicial. Determina um desvio de fase total para as funções de onda.*
- *<View>*
    - *Determina a razão entre a distância da câmera da origem e os vetores que definem os limites da tela. Desta forma, este parâmetro define o ângulo de abertura da câmera; colocando view pequeno, a câmera é mais fechada, colocando view grande, a câmera tende a ter um campo de visão muito maior.*

- *<xc>*
- *<yc>*
- *<zc>*
  - *Estas três coordenadas simplesmente definem a posição da câmera, que pode ser escolhida livremente em teoria. Na prática, é conveniente sempre escolher* $x >> y$ *no caso do desenho com triângulos, pois os triângulos são desenhados ao longo de linhas de x constante e com y crescente. Se x não for muito maior do que y, existe a possibilidade de oclusão de triângulos – eventualmente, um triângulo A que não seria visível ao observador pode ser desenhado sobre o triângulo B que bloqueia a visão do observador do triângulo A, causando um problema na imagem. O modo de renderização pixel-a-pixel não sofre deste defeito e resulta em uma imagem perfeita. Note que xc e yc não podem ser zero simultaneamente!*
- *<xcen1>*
- *<ycen1>*
  - *Estas duas coordenadas definem a posição do centro da onda 1, como indicado tanto em 5.1 quanto em 5.2.*
- *<w1>*
  - *Este parâmetro indica a variação senoidal da onda ao longo de variações de* $x^2 + y^2$ *ou de sua raiz quadrada. O usuário pode acertar este parâmetro exatamente com o que quiser que ele represente; de qualquer forma, este parâmetro se relaciona intimamente com o comprimento de onda. Valores extremos deste parâmetro só podem ser representados com a renderização pixel-a-pixel.*
- *<coeficiente de atenuação 1 , como em B de 5.1>*
  - *Este coeficiente determina como a função de onda cai em intensidade. A escolha deste parâmetro é importante para determinar a localidade do fenômeno de interferência e facilitar a sua visualização; a sua aplicação imediata não é tão quantitativa, de qualquer forma. O usuário pode experimentar valores quaisquer que atendam às suas necessidades.*
- *<amplitude 1, como em A de 5.1>*

- *Parâmetro bastante auto-explicativo que determina a amplitude da onda. As amplitudes podem ser variadas independentemente, visando a representação de uma interferência entre ondas bastante distintas. Note que valores extremos de amplitude só podem ser bem representados usando a renderização pixel-a-pixel.*
- *<fase 1, como ϕ de 5.1>*
    - *Este parâmetro proporciona um desvio de fase para a função de onda 1, exatamente como indicado em 5.1.*
- *<xcen2>*
- *<ycen2>*
- *<w2, como em 5.1>*
- *<coeficiente de atenuação 2 – como em B de 5.1>*
- *<amplitude 2, como em A de 5.1>*
- *<fase 2, como ϕ de 5.1>*
    - *Estes parâmetros são idênticos aos já explicados, exceto pelo fato de que estes referem à segunda onda, que será então sobreposta à primeira.*
- *<xL> (coordenadas para fonte de luz)*
- *<yL>*
- *<zL>*
    - *As coordenadas da fonte de luz serão usadas posteriormente para todas as rotinas subseqüentes de iluminação. As coordenadas da fonte de luz não podem se aproximar muito da função de onda sobreposta, pois isso causará divisão por zero!*
- *<lowx>*
- *<highx>*
- *<lowy>*
- *<highy>*
    - *Estes parâmetros muito relevantes dizem ao programa a faixa de x e y que deve ser calculada da superfície, quando se faz uso dos módulos que desenham por triângulos. Estes parâmetros não devem ser escolhidos de forma tal que o valor mínimo em uma coordenada se aproxime muito do*

valor máximo para esta coordenada, pois os triângulos são calculados com base nestes valores, e triângulos muito pequenos podem levar a uma divisão por zero, se o computador não tiver como distinguir um número muito próximo de zero (para o módulo do vetor normal do triângulo!) do número zero.

- *<x1 do gráfico>*
- *<y1 do gráfico>*
- *<x2 do gráfico>*
- *<y2 do gráfico>*
    - *Estes valores determinam a posição x e y dos pontos que definem o segmento ao qual a função de onda estará restrita para o desenho do gráfico, como indicado na figura 6. As coordenadas são significativas para o usuário apenas, não para o programa, e devem se relacionar com os w1 e w2 das ondas e com os valores lowx, highx, lowy e highy.*
- *<escala do gráfico da função>*
    - *Este valor determina a relação entre uma unidade de altura da função de onda e a quantidade de pixels utilizada para representar esta altura no gráfico. Desta forma, se a função f(x,y) tiver valor 1,5 e este fator de escala valer 100, o gráfico será tal que este ponto será representado 150 pixels acima do eixo x. O gráfico da função de onda será desenhado com uma curva branca.*
- *<escala do gráfico do quadrado da função>*
    - *Este valor determina a mesma relação pixels/altura do fator de escala já explicado, mas para o quadrado da função de onda, representado em uma tonalidade acinzentada.*
- *<resolução de renderização>*
    - *Este parâmetro será explicado em uma seção própria.*

O Apêndice A deste relatório permite a compreensão exata destas variáveis. O último parâmetro, resolução de renderização, será explicado em uma seção posterior deste mesmo relatório.

Note que todos os parâmetros podem ser dados em unidades significativas ao usuário, e o programa calculará tudo em termos destas. Isto deve ser salientado, pois denota a característica genérica deste programa.

Também foi dada a opção ao usuário de colocar um anteparo em uma dada posição (determinada entre os parâmetros, como já mencionado) e então o programa desenhará em um pequeno gráfico no canto superior direito da tela a restrição da superposição das funções de onda neste anteparo, assim como o seu quadrado. O gráfico tem limites dados por uma pequena escala na tela, sendo esta também dependente da entrada do usuário para os fatores de escala específicos – foram implementados dois, um para a função de onda e outro para o seu quadrado. Desta forma, dependendo das configurações do programa, deve ser possível exemplificar visualmente a interferência em um sistema de duas fendas através da visualização dos padrões gerados neste anteparo. Isto será discutido em uma seção a seguir.

Ainda em relação ao programa, o desenvolvimento de um módulo de animação foi algo que foi tentado extensivamente, mas o custo computacional envolvido em termos de cálculos e acesso de memória mostrou que a exibição de animações em tempo real é impossível; os computadores são muito lentos para isso. Um modo adicional de exibição de imagem ao longo de um período de oscilação foi implantado, de qualquer forma, mas não há exibição rápida da animação nos computadores testados até agora. Uma implementação apropriadamente de uma animação iria requerer o uso de rotinas de programação gráfica mais sofisticadas do que o escopo deste projeto sugeriria inicialmente.

6. Renderização Pixel-a-Pixel

Um desenvolvimento gráfico posterior permitiu que fosse criado um outro modo de representação da superfície, mais demorado mas muito mais representativo e sofisticado. Utilizou-se um procedimento que varre a tela pixel por pixel e determina, para cada um dos pixels, qual é exatamente o ponto de intersecção com a função de onda e qual é exatamente a tonalidade deste ponto. A varredura pixel a pixel é demorada na implementação atual do programa, mas resulta em uma imagem que é adequada à representação de qualquer

situação, mesmo aquelas nas quais os valores w1 e w2 são extremos um em relação ao outro, e aquelas nas quais os valores de amplitude são demasiadamente grandes. Quaisquer outras situações podem ser representadas na máxima qualidade de imagem possível no monitor em uso; porém, os cálculos associados são bastante demorados, como seria de se esperar. Centenas de milhares de pixels devem ser varridos para chegar na imagem final.

A varredura é feita considerando-se que cada pixel no monitor representa uma reta no espaço tridimensional, sendo que esta contém todos os pontos que possivelmente seriam representados por aquele pixel. A primeira intersecção desta reta com a função de onda somada é aquela que define o ponto que o observador verá; desta forma, o que o programa deve fazer é varrer esta reta começando de um ponto próximo ao observador. A condição que define se o ponto atual da reta é o que se deseja – que o obervador verá – é dada por 6.1.

$$f(x, y) - z \approx 0 \quad (6.1)$$

Resolver a equação 6.1 numericamente para x, y e z é portanto nosso objetivo. A implementação atual do programa considera simplesmente uma varredura em incrementos dados pelo usuário; isto pode ser otimizado muito ainda. O programa encerra a varredura da reta atual se encontrar uma condição próxima de 6.1.

Para a determinação adequada da cor que deve ser atribuída a cada pixel de 6.1, consideramos que a normal da função de onda em um dado ponto pode ser dada simplesmente pelo seu gradiente, definido por 6.2:

$$\left(-\frac{\partial f}{\partial x}, -\frac{\partial f}{\partial y}, 1\right) \quad (6.2)$$

Note a escolha de sinal de 6.2: a normal à superfície que desejamos é aquela que aponta para cima (z>0) e não para baixo. Tomando o produto escalar desta normal com o vetor unitário que define a direção da fonte de luz, pode-se obter, como antes, o co-seno do ângulo que representa a incidência de luz sobre a superfície neste ponto. A mesma função

de iluminação *L* utilizada antes com o procedimento *Illumina* pode então ser utilizada no cálculo da cor do pixel atual.

Um método de cálculo numérico para encontrar zeros de funções poderia ser aplicado a 6.1 para a aceleração deste processo de cálculo; note que este módulo de desenho pixel-a-pixel nem sequer estava incluído no projeto inicial e pode ser encarado como um extra. O aluno não teve tempo de aprimorar este módulo para a velocidade, mas o módulo de desenho pixel-a-pixel se encontra perfeitamente funcional de qualquer forma.

7. Aplicação Didática: interferência de fendas duplas

A utilidade do programa desenvolvido é muito grande devido à adaptabilidade de seus parâmetros à necessidade do usuário. Por exemplo, poderíamos tomar uma dada introdução à interferência de ondas como em um livro de nível secundário – colegial – e reescrever este trecho em particular deste livro utilizando apenas imagens desenvolvidas pelo nosso programa; isto notando-se ainda que este é um uso bastante qualitativo do programa ainda. O apêndice B denota como uma introdução à interferência poderia ser escrita usando exclusivamente imagens geradas pelo nosso programa.

De crucial importância nesta introdução são os parâmetros das imagens geradas. Note, por exemplo, como a figura 12-119 (a), que mostra uma figura de interferência, foi concebida: uma visão aérea de interferência entre duas ondas. Isto é perfeitamente possível com o nosso programa, bastando-se para tanto escolher x = y = 0 e z algum valor não muito pequeno.

Figura 7: Interferência entre duas ondas visualizada. Imagem obtida com renderização pixel-a-pixel, resolução 300.

Uma figura muito informativa, por exemplo, poderia ter sido obtida considerando-se meramente dois centros próximos um do outro e oferecendo ao usuário uma visão de cima desta imagem. Tipicamente, assim, seria possível visualizar as linhas nodais em apenas uma rápida observação da imagem subseqüente, como na figura 7. Os parâmetros específicos utilizados foram $x_{cen1} = -x_{cen2} = 1,5$; $y_{cen1} = y_{cen2} = -3$ para as posições dos centros, $w_1 = w_2 = 15$ para os análogos ao comprimento de onda de 5.1, e posição de câmera dada por (0,1,8).

A figura das franjas de interferência poderia ter sido gerada simplesmente com o nosso anteparo, que é desenhado sempre que se opta pela renderização triângulo-a-triângulo. Utilizando uma posição de câmera mais tradicional, mas qualquer, poderia se fazer uso das configurações utilizadas anteriormente para calcular uma imagem da função de onda restrita ao anteparo, assim como o seu quadrado. Isto pode ser visto na figura 8. Todos os coeficientes de atenuação foram zerados para a obtenção desta imagem.

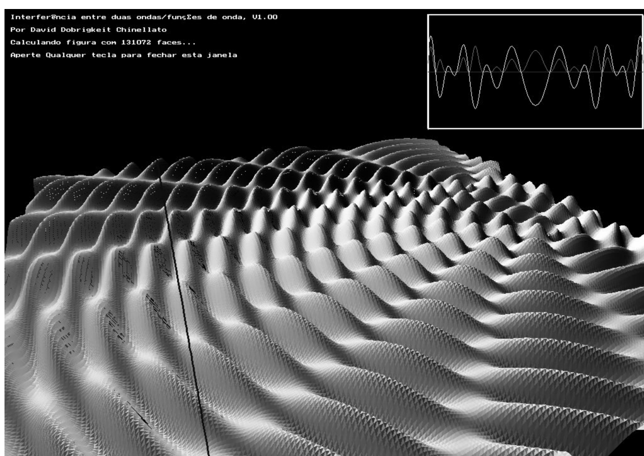

Figura 8: Interferência Entre duas ondas com corte.

De qualquer forma, estes são exemplos muito simples de todo o potencial do programa criado. Qualquer coisa relacionada com interferência de duas ondas bidimensionais é amplamente representável por este programa, tanto qualitativamente – que acreditamos que é um uso muito válido, bastando para isso observar a figura 7 – quanto quantitativamente, que é um uso válido, mas provavelmente menos comum.

8. Conclusão

Acreditamos que a versão atual do programa excede as capacidades originalmente planejadas do mesmo, já que este dispõe até mesmo de um módulo de renderização pixel-a-pixel capaz de gerar qualquer imagem, mesmo nas condições mais extremas, em qualidade limitada apenas pelo tempo que o usuário deseja esperar.

O módulo de desenho com triângulos também dispõe de recurso didático significativo com o desenho da função de onda restrita a um dado segmento de reta, como pode ser visto na figura 8.

Deve se ter em mente que este é apenas um exemplo que não é bem representativo da capacidade total do programa. Qualquer situação dada poderia ter sido representada,

mesmo nas condições extremas, com o renderizador pixel-a-pixel, que é tipicamente ótimo em qualidade de imagem, mas lento se comparado com a renderização pixel-a-pixel

Quanto à implementação de animação, um extra que poderia ter sido adicionado ao projeto, isto não foi feito devido à baixa velocidade de desenho de cada quadro usando ferramentas convencionais. Infelizmente, está fora do âmbito deste trabalho fazer uso de recursos gráficos extensivamente – isto nem sequer havia sido proposto inicialmente no projeto. Numerosas tentativas de se conseguir um resultado aceitável foram realizadas sem sucesso.

Acreditamos assim ter concluído o projeto inicialmente proposto e até mesmo adicionando fatores novos.

Apêndice A: *O Programa desenvolvido, visão passo a passo*

Será aqui reproduzido o programa utilizado para a concepção da figura 6 em sua totalidade. O programa está comentado e o código de cores aqui usado é bastante simples; em **verde** estão os comentários do programador (o aluno).

```
{=1.Início do programa=================================================}

program threedsurface;
uses crt,graph; {utiliza saída de terminal e módulo gráfico}

{=====================================================================}

{=2.Declaração de variáveis globais====================================}

var ma1,ma2,ma3,mb1,mb2,mb3,mc1,mc2,mc3:real;
    {corresponde a Alpha 1-2-3, Beta 1-2-3, Gama 1-2-3}
    xu,yu,zu,xw,yw,zw,xv,yv,zv:real; {coordenadas dos vetores u, v e w}
    xc,yc,zc,view,xL,yL,zL:real; {coordenadas da câmera,fonte de luz}
    gD,gM:integer; {referentes aos modos de tela do módulo graph}
    amp,coef,lowx,highx,lowy,highy,t:real;
    {parâmetros das ondas; parâmetros de desenho; índice de tempo}
    maxx,maxy:integer;
    {tamanho da tela na resolução do SO (sist. Operacional) utilizado}
    ccon1:integer;
    Scale,Scale2:Real;
    {fator de reescala para pixels do gráfico}
    xcut1,xcut2,ycut1,ycut2:real;
    {localização da restrição para o desenho do gráfico}
    w1,w2,phi1,phi2:real;
    {parâmetros da função de onda}
    params:text;
    {de onde será lido tudo: todos os parâmetros do programa!}
    choice:integer;
    {escolha do menu do programa}
    scanres:integer;
```

```
    {resolução de varredura}
    frame:array[1..30,1..64,1..64,1..2,1..3] of pointtype;
    colormatrix:array[1..30,1..64,1..64,1..2] of integer;
    {referentes à animação de 30 frames}


{=========================================================================}

{=3.Declaração de tipos de variáveis=====================================}

type
    point3d=object    {definição de ponto no R3}
    x,y,z:real;
    end;

{Para definir um quanta de superfície – um triângulo – atribuímos a ele 3
vértices, pontos no R3, como já definido, para cada um dos espaços. Três
para a projeção, e três para o espaço "real" que se deseja representar.
Além disso, a definição abaixo também atribui a cada face um valor illum,
referente ao co-seno do ângulo entre vetor normal e incidência de luz, e
z, referente à profundidade em relação à câmera.}

    face=object
    p1,p2,p3,p1in3d,p2in3d,p3in3d:point3d;
    illum,z:real;
    end;

{A superfície que será renderizada de -4 a 4 nas coordenadas x e y será
dividida em 256x256x2 triângulos. Para a parte animada, serão
considerados apenas 64x64x2 triângulos, ou seja, 8192 triângulos}

    Xgridtype=array[1..64,1..64] of point3d;
    gridtype=array[1..256,1..256] of point3d;

{=========================================================================}

{=4.Declaração de funções e procedimentos================================}
```

```
{Esta próxima função é a função de onda para as duas ondas utilizada!
Altamente relevante!!}

function f1(x,y,a1,amplit:real):real;
begin
     f1:=amplit*(Sin(w1*Sqrt(x*x+y*y)-phi1))*Exp(-coef*Sqrt(x*x+y*y));
end;

function f2(x,y,a1,amplit:real):real;
begin
     f2:=amplit*(Sin(w2*Sqrt(x*x+y*y)-phi2))*Exp(-coef*Sqrt(x*x+y*y));
end;

{esta função usa o princípio de superposição entre duas ondas do tipo
f1+f1 como definida anteriormente, mas com parâmetros levemente
modificados.}

function f(x,y,a1:real):real;
begin
        f:=f1(x-xcen1,y-ycen1,a1,amp)+f2(x-xcen2,y-ycen2,a1,amp2);
end;

{Esta próxima função diz como a superfície se comporta em relação ao
fluxo de luz incidente nela... por unidade de área! Isso é bastante
importante para a aparência final da superfície; em versões futuras,
podemos até mesmo pensar em mudar isso aqui... coisas interessantes são
possíveis!}

function L(param:real):integer;
begin
     If param>-1 then
l:=Trunc(255*(Sin((1/2)*(param+1)*Pi/2))*(Sin((1/2)*(param+1)*Pi/2)))
     else l:=0;
end;

{Vamos definir uma função/procedimento produto vetorial!}

procedure veprod(x1,y1,z1,x2,y2,z2:real;var x3,y3,z3:real);
```

```
var x3t,y3t,z3t:real;
begin

     x3t:=y1*z2-y2*z1;
     y3t:=z1*x2-z2*x1;
     z3t:=x1*y2-x2*y1;
     x3:=x3t;y3:=y3t;z3:=z3t;
end;
```

{Essa é outra trivial da álgebra linear: fazer as coisas virarem vetores unitários. Se algo vai dar problema, é aqui; é muito arriscado dividir por coisas que possivelmente estão perto de zero! Daria pra colocar um mecanismo de controle (se temp<10E-12, por exemplo, pare programa e diga pro usuário "ih, divisão por zero... ou quase! Mude parâmetros...")}

```
procedure UnitMod(var x1,y1,z1:real);
var temp:real;
begin
     temp:=Sqrt(x1*x1+y1*y1+z1*z1);
     x1:=x1/temp;y1:=y1/temp;z1:=z1/temp;
end;
```

{este próximo procedimento é aquele que foi explicado extensivamente na seção 4 do relatório; não devem haver grandes surpresas aqui. É basicamente o cálculo do fluxo por unidade de área de luz incidente, a partir do qual vai ser encontrada a cor atribuída ao triângulo!}

```
function Illumina(face1:face):real;
var xdl,ydl,zdl:real;
    xcen,ycen,zcen:real;
    xn,yn,zn:real;
begin
     xcen:=(face1.p1in3d.X+face1.p2in3d.X+face1.p3in3d.X)/3;
     ycen:=(face1.p1in3d.Y+face1.p2in3d.Y+face1.p3in3d.Y)/3;
     zcen:=(face1.p1in3d.Z+face1.p2in3d.Z+face1.p3in3d.Z)/3;
     xdl:=xL-xcen;ydl:=yL-ycen;zdl:=zL-zcen;
     UnitMod(xdl,ydl,zdl);
     Veprod(face1.p1in3d.X-face1.p2in3d.X,
```

```
            face1.p1in3d.Y-face1.p2in3d.Y,
            face1.p1in3d.Z-face1.p2in3d.Z,
            face1.p3in3d.X-face1.p2in3d.X,
            face1.p3in3d.Y-face1.p2in3d.Y,
            face1.p3in3d.Z-face1.p2in3d.Z,
            xn,yn,zn);
     UnitMod(xn,yn,zn);
     Illumina:=xdl*xn+ydl*yn+zdl*zn;
end;
```

{Uma outra bonita, simples e tranqüila: deslocamento em relação à câmera. Este procedimento simplesmente coloca a câmera na origem das coisas, o que é relevante ao cálculo de Dimkill e Morph.}

```
procedure dloc(var x,y,z:real);
    begin
    x:=x-xc; y:=y-yc; z:=z-zc;
    end;
```

{Muita álgebra linear aqui; trata-se da matriz de mudança de base. Todos os vetores foram normalizados, tudo convenientemente editado... as explicações para essa parte estão na seção 3 do relatório.}

```
procedure calcmat;
    var A: real;
    begin
         A:=(xu*zv*yw)-(yu*zv*xw)+(yu*xv*zw)-(xu*yv*zw);

         ma1:=((zv*yw)-(yv*zw))/A;
         mb1:=(yu*zw)/A;
         mc1:=(-yu*zv)/A;

         ma2:=((xv*zw)-(zv*xw))/A;
         mb2:=(-xu*zw)/A;
         mc2:=(xu*zv)/A;

         ma3:=((yv*xw)-(xv*yw))/A;
```

```
        mb3:=((xu*yw)-(yu*xw))/A;
        mc3:=((yu*xv)-(xu*yv))/A;
    end;
```

{este próximo procedimento calcula o vetor V, vertical em relação à exibição no monitor, de acordo com regras básicas de álgebra linear. Note que uma coisa bastante interessante é definir o ângulo de abertura da câmera, o que controla o efeito de perspectiva muito bem; para tanto, basta que o usuário entre com um parâmetro – view, neste caso – que é a tangente do ângulo de abertura vertical da câmera. Considerando então que o vetor V tem o seu módulo controlado por esta tangente de maneira trivial e que a câmera sempre observa o "alvo" origem, é possível calcular V da forma como este procedimento o faz!}

```
procedure findV;
    var t1,t2:real;
    begin
        t1:=Sqrt((xw*xw)+(yw*yw));
        t2:=Sqrt(view);
        xv:=(-((t2)*xw*zw)/t1);
        yv:=(-((t2)*yw*zw)/t1);
        zv:=((t2)*(t1));
    end;
```

{este outro procedimento acha o vetor U, horizontal em relação à projeção bidimensional e em relação ao monitor. Ele simplesmente toma o produto vetorial (é, dava para ter usado a definição de Veprod, mas não o fiz...) entre os vetores câmera-até-origem e V.}

```
procedure findU;
    var modU,modV,t:real;
    begin
        modV:=Sqrt((xv*xv)+(yv*yv)+(zv*zv));
        modU:=(4*modV)/3;

        xu:=((yw*zv)-(yv*zw));
        yu:=((xv*zw)-(zv*xw)); {Antes da correção modular!}
        zu:=((xw*yv)-(xv*yw));
```

```
        t:=-ModU/Sqrt((xw*xw*((yv*yv)+(zv*zv))+(zv*yw-yv*zw)*(zv*yw-
yv*zw)
           -2*xv*xw*(yv*yw+zv*zw)+(xv*xv)*((yw*yw)+(zw*zw))));

        xu:=xu*t;
        yu:=yu*t;
        zu:=zu*t;
    end;
```

{Essa é boa! Mudança de base!!! Note que trata-se apenas de uma multiplicação por matriz de mudança de base obtida com CalcMat!}

```
procedure Morph(var x,y,z:real);
    var xm,ym,zm:real;
    begin
        xm:=((ma1*x)+(ma2*y)+(ma3*z)); ym:=((mb1*x)+(mb2*y)+(mb3*z));
zm:=((mc1*x)+(mc2*y)+(mc3*z));
        x:=xm;y:=ym;z:=zm;
    end;
```

{Essa próxima, Dimkill, torna uma das dimensões finalmente obsoletas e considera que o que realmente queremos é uma projeção com efeito de perspectiva real, como se o observador estivesse a uma distância finita do objeto observado - e aliás, como se soubéssemos esta posição!}

```
procedure Dimkill(var x1,y1:real;z1:real);
var c1,c2:real;
    begin
        If (z1>0) then
          Begin
            c1:=(maxx/2)*(x1/z1);
            c2:=(maxy/2)*(y1/z1);
            x1:=c1; y1:=c2;
          End;
    End;
```

{agora vem a definição de superfície}

```
var sur:array[1..255,1..255,1..2] of face;
```

{isso é interessante: 256x256 quadrados, cada um deles com dois triângulos, é claro! Ou seja, 256x256x2 = 131072 triângulos...}

```
    grid:gridtype;
```

{esta última é a grade de pontos; isto inclui todos os vértices de todos os triângulos que serão utilizados... a variável grid deve, portanto, armazenar 65 mil pontos...}

{em seguida, o próximo passo seria localizar os pontos da variável GRID; faremos isso com a função LocatePoints.}

```
procedure locatepoints;
var c1,c2:integer;
begin
     for c1:=1 to 256 do
         for c2:=1 to 256 do
             begin
             grid[c1,c2].X:=lowx+((c1-1)/255)*(highx-lowx);
             grid[c1,c2].Y:=lowy+((c2-1)/255)*(highy-lowy);
             grid[c1,c2].Z:=f(grid[c1,c2].X,grid[c1,c2].Y,t);
             end;

end;
```

{versão de poucos polígonos: 64x64x2 triângulos apenas}

```
procedure Xlocatepoints;
var c1,c2:integer;
begin
     for c1:=1 to 64 do
         for c2:=1 to 64 do
             begin
             Xgrid[c1,c2].X:=lowx+((c1-1)/63)*(highx-lowx);
             Xgrid[c1,c2].Y:=lowy+((c2-1)/63)*(highy-lowy);
```

```
            Xgrid[c1,c2].Z:=f(Xgrid[c1,c2].X,Xgrid[c1,c2].Y,t);
            end;

end;
```

{este próximo procedimento também é bastante importante. Utilizando as versões projetadas dos pontos da grade GRID, vamos atribuir agora a cada triângulo ("face" como definida anteriormente) os seus vértices e já vamos até calcular os fatores de iluminação com a nossa função Illumina! Note que devemos ainda estar no espaço NÃO-PROJETADO para usar esta função; isso quer dizer que ela deve ser aplicada ANTES do procedimento "Morphingtime",claro!}

```
procedure suratt3D;
var c1,c2:integer;
begin
     for c1:=1 to 255 do
         for c2:=1 to 255 do
             begin
             sur[c1,c2,1].p1in3d:=grid[c1,c2];
             sur[c1,c2,1].p2in3d:=grid[c1,c2+1];
             sur[c1,c2,1].p3in3d:=grid[c1+1,c2+1];
             sur[c1,c2,1].Illum:=Illumina(sur[c1,c2,1]);

             sur[c1,c2,2].p1in3d:=grid[c1,c2];
             sur[c1,c2,2].p2in3d:=grid[c1+1,c2+1];
             sur[c1,c2,2].p3in3d:=grid[c1+1,c2];
             sur[c1,c2,2].Illum:=Illumina(sur[c1,c2,2]);
              end;

end;

procedure Xsuratt3D;
var c1,c2:integer;
begin
     for c1:=1 to 63 do
         for c2:=1 to 63 do
             begin
```

```
            Xsur[c1,c2,1].p1in3d:=Xgrid[c1,c2];
            Xsur[c1,c2,1].p2in3d:=Xgrid[c1,c2+1];
            Xsur[c1,c2,1].p3in3d:=Xgrid[c1+1,c2+1];
            Xsur[c1,c2,1].Illum:=Illumina(Xsur[c1,c2,1]);

            Xsur[c1,c2,2].p1in3d:=Xgrid[c1,c2];
            Xsur[c1,c2,2].p2in3d:=Xgrid[c1+1,c2+1];
            Xsur[c1,c2,2].p3in3d:=Xgrid[c1+1,c2];
            Xsur[c1,c2,2].Illum:=Illumina(Xsur[c1,c2,2]);
             end;

end;


{O procedimento Morphingtime é responsável por projetar a grade definida
por grid no espaço projetado em duas dimensões! É bastante importante.}

procedure morphingtime;
var c1,c2:integer;
begin
     for c1:=1 to 256 do
         for c2:=1 to 256 do
            begin
            Dloc(grid[c1,c2].X,grid[c1,c2].Y,grid[c1,c2].Z);
            Morph(grid[c1,c2].X,grid[c1,c2].Y,grid[c1,c2].Z);
            Dimkill(grid[c1,c2].X,grid[c1,c2].Y,grid[c1,c2].Z);
            end;
end;

procedure Xmorphingtime;
var c1,c2:integer;
begin
     for c1:=1 to 64 do
         for c2:=1 to 64 do
            begin
            Dloc(Xgrid[c1,c2].X,Xgrid[c1,c2].Y,Xgrid[c1,c2].Z);
            Morph(Xgrid[c1,c2].X,Xgrid[c1,c2].Y,Xgrid[c1,c2].Z);
            Dimkill(Xgrid[c1,c2].X,Xgrid[c1,c2].Y,Xgrid[c1,c2].Z);
```

```
            end;
end;
```

{o próximo procedimento trabalha com os valores de posição na tela! Deve-se atentar à possibilidade de confundir posição no espaço original e projetado. Estes valores, dos vértices p1, p2 e p3, são posições no espaço projetado! (no monitor)}

```
procedure suratt;
var c1,c2:integer;
begin
     for c1:=1 to 255 do
         for c2:=1 to 255 do
             begin
             sur[c1,c2,1].p1:=grid[c1,c2];
             sur[c1,c2,1].p2:=grid[c1,c2+1];
             sur[c1,c2,1].p3:=grid[c1+1,c2+1];

             sur[c1,c2,2].p1:=grid[c1,c2];
             sur[c1,c2,2].p2:=grid[c1+1,c2+1];
             sur[c1,c2,2].p3:=grid[c1+1,c2];

             sur[c1,c2,1].z:=
             (grid[c1,c2].z+grid[c1,c2+1].z+grid[c1+1,c2+1].z)/3;
             sur[c1,c2,2].z:=
             (grid[c1,c2].z+grid[c1+1,c2+1].z+grid[c1+1,c2].z)/3;
             end;
end;

procedure Xsuratt;
var c1,c2:integer;
begin
     for c1:=1 to 63 do
         for c2:=1 to 63 do
             begin
             Xsur[c1,c2,1].p1:=Xgrid[c1,c2];
             Xsur[c1,c2,1].p2:=Xgrid[c1,c2+1];
             Xsur[c1,c2,1].p3:=Xgrid[c1+1,c2+1];
```

```
            Xsur[c1,c2,2].p1:=Xgrid[c1,c2];
            Xsur[c1,c2,2].p2:=Xgrid[c1+1,c2+1];
            Xsur[c1,c2,2].p3:=Xgrid[c1+1,c2];

            Xsur[c1,c2,1].z:=
            (Xgrid[c1,c2].z+Xgrid[c1,c2+1].z+Xgrid[c1+1,c2+1].z)/3;
            Xsur[c1,c2,2].z:=
            (Xgrid[c1,c2].z+Xgrid[c1+1,c2+1].z+Xgrid[c1+1,c2].z)/3;
            end;
end;
```

{este próximo procedimento é responsável por desenhar os triângulos na tela. Compreensão exata do procedimento pode ser obtida apenas com conhecimento da linguagem de programação utilizada; mas é bastante simples entender o funcionamento geral da coisa: define-se triângulos pequenos em memória, atribui-se cores e o desenho de fato ocorre com o procedimento FillPoly. É só isso, repetido para a superfície toda!}

```
procedure drawnow;
var c1,c2:integer;
    tri1,tri2:array[1..3] of pointtype;
begin

     for c1:=1 to 255 do
         for c2:=1 to 255 do
            begin

            tri1[1].x:=Trunc((maxx/2)+sur[c1,c2,1].p1.x);
            tri1[1].y:=Trunc((maxy/2)-sur[c1,c2,1].p1.y);
            tri1[2].x:=Trunc((maxx/2)+sur[c1,c2,1].p2.x);
            tri1[2].y:=Trunc((maxy/2)-sur[c1,c2,1].p2.y);
            tri1[3].x:=Trunc((maxx/2)+sur[c1,c2,1].p3.x);
            tri1[3].y:=Trunc((maxy/2)-sur[c1,c2,1].p3.y);
            SetColor(l(sur[c1,c2,1].illum));
            FillPoly(3,tri1);

            tri2[1].x:=Trunc((maxx/2)+sur[c1,c2,2].p1.x);
```

```
            tri2[1].y:=Trunc((maxy/2)-sur[c1,c2,2].p1.y);
            tri2[2].x:=Trunc((maxx/2)+sur[c1,c2,2].p2.x);
            tri2[2].y:=Trunc((maxy/2)-sur[c1,c2,2].p2.y);
            tri2[3].x:=Trunc((maxx/2)+sur[c1,c2,2].p3.x);
            tri2[3].y:=Trunc((maxy/2)-sur[c1,c2,2].p3.y);
            SetColor(l(sur[c1,c2,2].Illum));
            FillPoly(3,tri2);
            End;
End;
```

{O procedimento mostrado a seguir é responsável por alocar na memória todas as posições necessárias à efetuação da animação, mesmo que lenta, implementada atualmente. Idêntico ao Drawnow, exceto que a saída é na memória, não na tela!}

```
procedure drawnowinmemory(framenumber:integer);
var c1,c2:integer;
    tri1,tri2:array[1..3] of pointtype;
begin

     for c1:=1 to 63 do
         for c2:=1 to 63 do
             begin
             tri1[1].x:=Trunc((maxx/2)+Xsur[c1,c2,1].p1.x);
             tri1[1].y:=Trunc((maxy/2)-Xsur[c1,c2,1].p1.y);
             tri1[2].x:=Trunc((maxx/2)+Xsur[c1,c2,1].p2.x);
             tri1[2].y:=Trunc((maxy/2)-Xsur[c1,c2,1].p2.y);
             tri1[3].x:=Trunc((maxx/2)+Xsur[c1,c2,1].p3.x);
             tri1[3].y:=Trunc((maxy/2)-Xsur[c1,c2,1].p3.y);
             colormatrix[framenumber,c1,c2,1]:=l(Xsur[c1,c2,1].illum);
             frame[framenumber,c1,c2,1]:=tri1;

             tri2[1].x:=Trunc((maxx/2)+Xsur[c1,c2,2].p1.x);
             tri2[1].y:=Trunc((maxy/2)-Xsur[c1,c2,2].p1.y);
             tri2[2].x:=Trunc((maxx/2)+Xsur[c1,c2,2].p2.x);
             tri2[2].y:=Trunc((maxy/2)-Xsur[c1,c2,2].p2.y);
             tri2[3].x:=Trunc((maxx/2)+Xsur[c1,c2,2].p3.x);
             tri2[3].y:=Trunc((maxy/2)-Xsur[c1,c2,2].p3.y);
```

```
            colormatrix[framenumber,c1,c2,2]:=l(Xsur[c1,c2,2].Illum);
            frame[framenumber,c1,c2,2]:=tri2;
            End;
End;
```

{este procedimento mostrado anteriormente deve ser associado ao próximo, que faz as animações do começo ao fim!}

```
procedure OperationDrawAniminmem;
var curfram:integer;
begin
     camplace;t:=0;
     Writeln;
     Writeln('Você escolheu realizar uma animaçÃo em tempo real.');
     Writeln('Calculando 30 frames');
     For curfram:=1 to 30 do
     Begin
           Xlocatepoints;
           Xsuratt3d;
           Xmorphingtime;
           Xsuratt;
           drawnowinmemory(curfram);
           phi1:=phi1+(1/15)*Pi;
           phi2:=phi2+(1/15)*Pi;
           Write(curfram);
           Write('/');
     End;
     Writeln;
End;
```

{depois de desenhada na memória, as coisas têm de ser desenhadas na tela, certo? Sim, e é o que o próximo procedimento faz!}

```
procedure DrawAnimeNOW;
var curfram,c1,c2:integer;
        tri1:array[1..3] of pointtype;
begin
     gD:=Detect;gM:=GetMaxMode;
```

```
    Initgraph(gD,gM,'');
    If GraphResult<>grOK then Halt(1);
    maxx:=GetMaxX;
    maxy:=GetMaxY;
    For ccon1:=1 to 255 do
    SetRGBPalette(ccon1,ccon1,ccon1,ccon1);
    Moveto(10,10);
    Outtext('Animação!');
    Moveto(10,30);
    Outtext('Por David Dobrigkeit Chinellato');
    Repeat
    For curfram:=1 to 30 do
    Begin
       for c1:=1 to 63 do
               for c2:=1 to 63 do
                       begin
                       SetColor(colormatrix[curfram,c1,c2,1]);
                       FillPoly(3,frame[curfram,c1,c2,1]);
                       SetColor(colormatrix[curfram,c1,c2,2]);
                       FillPoly(3,frame[curfram,c1,c2,2]);
                       End;
                       delay(40);
                       ClearDevice;
    End;
    Until keypressed;
    CloseGraph;

End;

{algumas outras rotinas auxiliares definidas para a animação...}

procedure GetGraphParams;
begin
    gD:=Detect;gM:=GetMaxMode;
    Initgraph(gD,gM,'');
    If GraphResult<>grOK then Halt(1);
    maxx:=GetMaxX;
    maxy:=GetMaxY;
```

```
     delay(1);
     Closegraph;
End;

procedure Anime;
begin
        GetGraphParams;
        OperationDrawAniminMem;
        Writeln;
        Writeln('Tudo calculado sem erros.');
        DrawAnimeNOW;
end;
```

{Este próximo procedimento posiciona a câmera e é remanescente de uma animação de câmera que poderá ou não ser implementada no futuro. Problemas de oclusão devem ser corrigidos antes. Por enquanto, esta parte do programa é OBSOLETA! Note que os senos e cosenos de rotação da câmera NÃO REGISTRAM a passagem do tempo (os valores dos seus argumentos deveriam ser "t" para que as variações fossem registradas! Neste caso, a câmera apresentaria rotação ao redor do eixo z, mais uma curiosidade perfeitamente possível...}

```
procedure camplace;
begin
     xc:=Cos(0)*xc-Sin(0)*yc;
     yc:=Sin(0)*xc+Cos(0)*yc;
     xw:=-xc;
     yw:=-yc;
     FindV;FindU;CalcMat;
End;
```

Este procedimento desenha a linha de restrição da superposição das duas funções de onda sobre o desenho... Desta forma, é possível associar visualmente o gráfico do canto superior direito da tela a uma parte da função de onda bidimensional.}

```
procedure drawplaneofintersect;
```

```
var param:integer;
        Ax,Ay,Az:real;
        Ax2,Ay2,Az2:real;
        Axt,Ayt,Ax2t,Ay2t:integer;
begin
        SetColor(White);
        Ax2:=xcut1;
        Ay2:=ycut1;
        Az2:=f(Ax2,Ay2,t);
        SetLineStyle(0,0,3);
        for param:=1 to 199 do
                begin
                        Ax:=xcut1+((param-1)/200)*(xcut2-xcut1);
                        Ay:=ycut1+((param-1)/200)*(ycut2-ycut1);
                        Az:=f(Ax,Ay,t);
                        Ax2:=xcut1+(param/200)*(xcut2-xcut1);
                        Ay2:=ycut1+(param/200)*(ycut2-ycut1);
                        Az2:=f(Ax2,Ay2,t);
                        dloc(Ax,Ay,Az);
                        dloc(Ax2,Ay2,Az2);
                        Morph(Ax,Ay,Az);
                        Morph(Ax2,Ay2,Az2);
                        Dimkill(Ax,Ay,Az);
                        Dimkill(Ax2,Ay2,Az2);
                        Axt:=Trunc(maxx/2+Ax);
                        Ayt:=Trunc(maxy/2-Ay);
                        Ax2t:=Trunc(maxx/2+Ax2);
                        Ay2t:=Trunc(maxy/2-Ay2);
                        Moveto(Axt,Ayt);
                        Lineto(Ax2t,Ay2t);

                end;
                Moveto (Ax2t+5,Ay2t+5);
                SetColor(Black);
                Outtext('Plan. Inter.');

End;
```

```
{O procedimento a seguir é responsável por carregar do disco - especificamente, do arquivo params.txt - todos os parâmetros que definem como o programa será executado!}

procedure OpenParams;
var str0:string;
        real1:real;
        code:integer;
begin
        Assign(params,'params.txt');
        Reset(params);
        Readln(params,str0); Val(str0,real1,code);      t:=real1;
        Readln(params,str0); Val(str0,real1,code);      View:=real1;
        Readln(params,str0); Val(str0,real1,code);      xc:=real1;
        Readln(params,str0); Val(str0,real1,code);      yc:=real1;
        Readln(params,str0); Val(str0,real1,code);      zc:=real1;

        Readln(params,str0); Val(str0,real1,code);      xcen1:=real1;
        Readln(params,str0); Val(str0,real1,code);      ycen1:=real1;
        Readln(params,str0); Val(str0,real1,code);      w1:=real1;
        Readln(params,str0); Val(str0,real1,code);      coef:=real1;
        Readln(params,str0); Val(str0,real1,code);      amp:=real1;
        Readln(params,str0); Val(str0,real1,code);      phi1:=real1;


        Readln(params,str0); Val(str0,real1,code);      xcen2:=real1;
        Readln(params,str0); Val(str0,real1,code);      ycen2:=real1;
        Readln(params,str0); Val(str0,real1,code);      w2:=real1;
        Readln(params,str0); Val(str0,real1,code);      coef2:=real1;
        Readln(params,str0); Val(str0,real1,code);      amp2:=real1;
        Readln(params,str0); Val(str0,real1,code);      phi2:=real1;

        Readln(params,str0); Val(str0,real1,code);      xL:=real1;
        Readln(params,str0); Val(str0,real1,code);      yL:=real1;
        Readln(params,str0); Val(str0,real1,code);      zL:=real1;

        Readln(params,str0); Val(str0,real1,code);      lowx:=real1;
        Readln(params,str0); Val(str0,real1,code);      highx:=real1;
```

```
        Readln(params,str0); Val(str0,real1,code);       lowy:=real1;
        Readln(params,str0); Val(str0,real1,code);       highy:=real1;

        Readln(params,str0); Val(str0,real1,code);       xcut1:=real1;
        Readln(params,str0); Val(str0,real1,code);       ycut1:=real1;
        Readln(params,str0); Val(str0,real1,code);       xcut2:=real1;
        Readln(params,str0); Val(str0,real1,code);       ycut2:=real1;
        Readln(params,str0); Val(str0,real1,code);       Scale:=real1;
        Readln(params,str0); Val(str0,real1,code);       Scale2:=real1;

        Readln(params,str0); scanres:=StrToInt(str0);
        Close(params);
End;

{Este próximo procedimento altera qualquer um dos parâmetros. Isto é bom,
pois estes parâmetros serão todos salvados quando o programa for
fechado!}

procedure Changeparams;
var localint,localint2:integer;
begin
        ClrScr;
        Writeln('Você optou por alterar parâmetros do programa. Escolha
                  uma categoria:');
        Writeln;
        Writeln('1) Parâmetros das funções de onda');
        Writeln('   (inclui comprimentos de onda, coeficientes de
                     atenuação, amplitudes, etc.)');
        Writeln;
        Writeln('2) Parâmetros de desenho');
        Writeln('   (inclui posição da câmera, abertura de câmera,
                     posição de luz)');
        Writeln;
        Writeln('3) Parâmetros da função restrita a um segmento');
        Writeln('   (inclui escalas do gráfico e posição dos pontos que
                     definem o segmento)');
        Writeln;
        Writeln('4) Nenhum deles; volte ao menu principal do programa');
```

```
Writeln;
Write('Sua Escolha: ');
Readln(localint);
If localint=1 then
Begin
        ClrScr;
        Writeln('Escolha Exatamente qual parâmetro deseja
                alterar: ');
        Writeln;
        Writeln('1) Posição X do centro da Onda 1');
        Writeln('2) Posição Y do centro da Onda 1');
        Writeln('3) Frequência Angular da Onda 1');
        Writeln('4) Coeficiente de Atenuação da Onda 1');
        Writeln('5) Amplitude da Onda 1');
        Writeln('6) Fase da Onda 1');
        Writeln;
        Writeln('7) PosiçÃo X do centro da Onda 2');
        Writeln('8) PosiçÃo Y do centro da Onda 2');
        Writeln('9) Frequência Angular da Onda 2');
        Writeln('10) Coeficiente de AtenuaçÃo da Onda 2');
        Writeln('11) Amplitude da Onda 2');
        Writeln('12) Fase da Onda 2');
        Writeln('(qualquer outro valor retorna ao menu
                      principal)');
        Writeln;
        Write('Sua Escolha: ');
        Readln(localint2);
        If localint2=1 then
          Begin
          Writeln;
          Write('Novo valor: ');
          Readln(xcen1);
          End;
        If localint2=2 then
          Begin
          Writeln;
          Write('Novo valor: ');
          Readln(ycen1);
```

```
  End;
If localint2=3 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(w1);
  End;
If Localint2=4 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(coef);
  End;
If localint2=5 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(amp);
  End;
If localint2=6 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(phi1);
  End;

If localint2=7 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(xcen2);
  End;
If localint2=8 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(ycen2);
  End;
```

```
        If localint2=9 then
          Begin
          Writeln;
          Write('Novo valor: ');
          Readln(w2);
          End;
        If localint2=10 then
          Begin
          Writeln;
          Write('Novo valor: ');
          Readln(coef2);
          End;
        If localint2=11 then
          Begin
          Writeln;
          Write('Novo valor: ');
          Readln(amp2);
          End;
        If localint2=12 then
          Begin
          Writeln;
          Write('Novo valor: ');
          Readln(phi2);
          End;
   Writeln('Valor Alterado.');
End;
If LocalInt=2 then
Begin
        ClrScr;
        Writeln('Escolha Exatamente o parâmetro de desenho que
                pretende alterar:');
        Writeln;
        Writeln('1) Parâmetro view (tangente do ângulo de
                      abertura da câmera)');
        Writeln('2) Posição X da câmera');
        Writeln('3) Posição Y da câmera');
        Writeln('4) Posição Z da câmera');
        Writeln('5) Valor limite para desenho da superfície:
```

```
                    menor X');
Writeln('6) Valor limite para desenho da superfície:
                    maior X');
Writeln('7) Valor limite para desenho da superfície:
                    menor Y');
Writeln('8) Valor limite para desenho da superfície:
                    maior Y');
Writeln('9) PosiçÃo X da luz');
Writeln('10) PosiçÃo Y da luz');
Writeln('11) PosiçÃo Z da luz');
Writeln('12) ResoluçÃo de renderizaçÃo');
Writeln('(qualquer outro valor retorna ao menu
              principal)');
Writeln;
Write('Sua Escolha: ');
Readln(localint2);
If Localint2=1 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(view);
  End;
If Localint2=2 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(xc);
  End;
If Localint2=3 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(yc);
  End;
If Localint2=4 then
  Begin
  Writeln;
  Write('Novo valor: ');
```

```
  Readln(yc);
  End;
If Localint2=5 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(lowx);
  End;
If Localint2=6 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(highx);
  End;
If Localint2=7 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(lowy);
  End;
If Localint2=8 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(highy);
  End;
If Localint2=9 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(xL);
  End;
If Localint2=10 then
  Begin
  Writeln;
  Write('Novo valor: ');
  Readln(yL);
  End;
```

```
        If Localint2=11 then
          Begin
          Writeln;
          Write('Novo valor: ');
          Readln(zL);
          End;
        If LocalInt2=12 then
          Begin Writeln;
          Writeln('O processo já é lento; nÃo coloque valores
                      muito grandes!');
          Writeln('O resultado ficará muito bom, mas demorar
                      muito tempo!!!');
          Write('Novo valor: ');
          Readln(scanres);
          End;
        Writeln('Valor Alterado.');
End;

If Localint=3 then
Begin
        ClrScr;
        Writeln('Você optou por alterar um parâmetro relacionado
          ao gráfico da função de onda restrita a um segmento.');
        Writeln;
        Writeln('1) Valor X do primeiro ponto do segmento');
        Writeln('2) Valor Y do primeiro ponto do segmento');
        Writeln('3) Valor X do segundo ponto do segmento');
        Writeln('4) Valor Y do segundo ponto do segmento');
        Writeln('5) Fator de escala (pixels/unidade de altura)
               para função de onda restrita');
        Writeln('6) Fator de escala (pixels/unidade de altura)
               para quadrado da funçÃo de onda');
        Writeln;
        Write('Sua escolha: ');
        Readln(localint2);
        If localint2=1 then Begin
          Writeln;
          Write('Novo valor: ');
```

```
                  Readln(xcut1);
                  End;
                If localint2=2 then
                  Begin
                  Writeln;
                  Write('Novo valor: ');
                  Readln(ycut1);
                  End;
                If localint2=3 then
                  Begin
                  Writeln;
                  Write('Novo valor: ');
                  Readln(xcut2);
                  End;
                If localint2=4 then
                  Begin
                  Writeln;
                  Write('Novo valor: ');
                  Readln(ycut2);
                  End;
                If localint2=5 then
                  Begin
                  Writeln;
                  Write('Novo valor: ');
                  Readln(Scale);
                  End;
                If localint2=6 then
                  Begin
                  Writeln;
                  Write('Novo valor: ');
                  Readln(Scale2);
                  End;
                Writeln('Valor Alterado.');
        End;
End;
```

```
{não adianta nada oferecer a escolha de mudança de parâmetros se o
programa não se lembrar das mudanças, certo? Então ele as salvará
convenientemente!!! A seguir, no próximo procedimento!}

procedure XExit;
Begin
        Writeln;
        Writeln('Salvando parƒmetros.');
        Assign(params,'params.txt');
        Rewrite(params);
        Writeln(params,t);              Write('.');
        Writeln(params,view);           Write('.');
        Writeln(params,xc);             Write('.');
        Writeln(params,yc);             Write('.');
        Writeln(params,zc);             Write('.');
        Writeln(params,xcen1);          Write('.');
        Writeln(params,ycen1);          Write('.');
        Writeln(params,w1);             Write('.');
        Writeln(params,coef);           Write('.');
        Writeln(params,amp);            Write('.');
        Writeln(params,phi1);           Write('.');
        Writeln(params,xcen2);          Write('.');
        Writeln(params,ycen2);          Write('.');
        Writeln(params,w2);             Write('.');
        Writeln(params,coef2);          Write('.');
        Writeln(params,amp2);           Write('.');
        Writeln(params,phi2);           Write('.');
        Writeln(params,xL);             Write('.');
        Writeln(params,yL);             Write('.');
        Writeln(params,zL);             Write('.');
        Writeln(params,lowx);           Write('.');
        Writeln(params,highx);          Write('.');
        Writeln(params,lowy);           Write('.');
        Writeln(params,highy);          Write('.');
        Writeln(params,xcut1);          Write('.');
        Writeln(params,ycut1);          Write('.');
        Writeln(params,xcut2);          Write('.');
        Writeln(params,ycut2);          Write('.');
```

```
        Writeln(params,scale);          Write('.');
        Writeln(params,scale2);         Write('.');
        Writeln(params,scanres);        Write('.');
        Close(params);
        Writeln;
        Writeln;
        Writeln('Obrigado por usar este programa.');
        Writeln('Por David D. Chinellato, Junho de 2004');
        Exit;
End;


{este próximo procedimento é o crucial para a renderização pixel-a-pixel.
Note como os índices c1 e c2 efetuam a varredura na tela, e c3 efetua
varredura ao longo de uma reta definida por um mesmo pixel na tela. Esta
parte está passível de mudanças para otimização de velocidade do programa
ainda!}

var screen:array[-700..700,-500..500] of integer;

procedure ScanLineRender(maxxscan,maxyscan:integer);
var c1,c2,c3:integer;
    count,count1:longint;
    xscan,yscan,zscan:real;
    Xsc,Ysc,Zsc:real;
    dfdx,dfdy:real;
    Xnorm,Ynorm,Znorm:real;
    xdl,ydl,zdl:real;
begin
        count:=0;
        count1:=0;
    for c1:=-maxxscan to maxxscan do
        for c2:=-maxyscan to maxyscan do
                Begin
                {definir atual vetor de scan!!!}
                FindV;
                FindU;
                xscan:=(-xc+(c1/maxxscan)*xu+(c2/maxyscan)*xv)/scanres;
```

```
                yscan:=(-yc+(c1/maxxscan)*yu+(c2/maxyscan)*yv)/scanres;
                zscan:=(-zc+(c1/maxxscan)*zu+(c2/maxyscan)*zv)/scanres;
                Xsc:=xc;Ysc:=yc;Zsc:=zc;
                c3:=1;
                Repeat
                c3:=c3+1;
                Xsc:=xsc+xscan;
                Ysc:=ysc+yscan;
                Zsc:=zsc+zscan;
                If  (f(xsc,ysc,t)-zsc)>0 then
                {isso , indicativo de que entrei na superfície....}
                {preciso agora calcular a normal!}
                Begin
                        dfdx:=200*(f(xsc+1/200,ysc,t)-f(xsc,ysc,t));
                        dfdy:=200*(f(xsc,ysc+1/200,t)-f(xsc,ysc,t));
                        xnorm:=-dfdx;ynorm:=-dfdy;znorm:=+1;
                        UnitMod(Xnorm,Ynorm,Znorm);
                        xdl:=xL-xsc;ydl:=yL-ysc;zdl:=zL-zsc;
                        UnitMod(xdl,ydl,zdl);
                        screen[c1,c2]:=l(xnorm*xdl+ynorm*ydl+znorm*zdl);
                        c3:=3*scanres+1;
                        count:=count+1;
                        If count-count1>999 then Begin
Writeln(count);count1:=count;End;
                End;
                Until c3>3*scanres;
                End;
End;

{para efetuar a renderização pixel-a-pixel, ainda falta a rotina de
desenho dos pixels calculados aqui! Veja:}

procedure Scanlinetodeath;
var c1,c2:integer;
        maxxscan,maxyscan:integer;
begin
     gD:=Detect;gM:=GetMaxMode;
     Initgraph(gD,gM,'');
```

```
     If GraphResult<>grOK then Halt(1);
     maxx:=GetMaxX;
     maxy:=GetMaxY;
     maxxscan:=Trunc(maxx/2);
     maxyscan:=Trunc(maxy/2);
     For ccon1:=1 to 255 do
     SetRGBPalette(ccon1,ccon1,ccon1,ccon1);
     Moveto(10,10);
     Outtext('Render - pixel a pixel!!!');
     Moveto(10,30);
     Outtext('Por David Dobrigkeit Chinellato');
     ScanlineRender(maxxscan,maxyscan);
     Readln;
     For c1:=-maxxscan to maxxscan do
     For c2:=-maxyscan to maxyscan do
     Begin
        Putpixel(maxxscan+c1,maxyscan-c2,screen[c1,c2]);
     End;
     Readln;
     CloseGraph;
End;
```

{Este próximo procedimento desenha o gráfico da restrição da função de onda total no canto superior direito do monitor. Além disso, ele posiciona eixos, mas nada de escalas ainda... O procedimento também desenha o quadrado da função de onda total, mas com outro fator de escala.}

```
procedure drawGraph;
var horsize,versize:integer;
        counter:integer;
        fsquare1,fsquare2:real;
begin
        horsize:=Trunc(maxx/3);
        versize:=Trunc(maxy/4);
        SetColor(255);
        SetFillStyle(SolidFill,White);
        Rectangle(maxx-10,10,maxx-10-horsize,10+versize);
```

```
        SetLineStyle(0,0,1);
        SetColor(55);
        Line(maxx-10-horsize,10+Trunc(versize/2),maxx-
                  10,10+Trunc(versize/2));
        SetColor(120);
        For counter:=1 to horsize do
        Begin
                fsquare1:=f(xcut1+((counter-1)/horsize)*(xcut2-xcut1),
                            ycut1+((counter-1)/horsize)*(ycut2-ycut1),t)*
                          f(xcut1+((counter-1)/horsize)*(xcut2-xcut1),
                            ycut1+((counter-1)/horsize)*(ycut2-ycut1),t);
                fsquare2:=f(xcut1+((counter/horsize)*(xcut2-xcut1)),
                            ycut1+((counter/horsize)*(ycut2-ycut1)),t)*
                          f(xcut1+((counter/horsize)*(xcut2-xcut1)),
                            ycut1+((counter/horsize)*(ycut2-ycut1)),t);
           Moveto(maxx-11-horsize+counter,10+Trunc(versize/2)-
                        Trunc(Scale2*fsquare1));
           Lineto(maxx-11-horsize+counter+1,10+Trunc(versize/2)-
                        Trunc(Scale2*fsquare2));
        End;
        SetColor(255);
        For counter:=1 to horsize do
        begin
           Moveto(maxx-11-horsize+counter,10+(Trunc(versize/2))-
            Trunc(Scale*f(xcut1+((counter-1)/horsize)*(xcut2-xcut1),
                          ycut1+((counter-1)/horsize)*(ycut2-ycut1),t)));
           Lineto(maxx-11-horsize+counter+1,10+(Trunc(versize/2))-
            Trunc(Scale*f(xcut1+((counter/horsize)*(xcut2-xcut1)),
                          ycut1+((counter/horsize)*(ycut2-ycut1)),t)));
        End;
End;

{Este próximo procedimento desenha o menu de opções!}

procedure DrawMenu;
begin
        ClrScr;
        Writeln('===================================');
```

```
        Writeln('Módulo de Ondulatória bidimensional');
        Writeln('===================================');
        Writeln;
        Writeln('Os parƒmetros utilizados anteriormente estÃo
            carregados.');
        Writeln('Entre com opção:');
        Writeln;
        Writeln('1) Desenhe a onda! Use triângulos para aproximar
            superfície.');
        Writeln('   (opção mais rápida)');
        Writeln;
        Writeln('2) Alterar parâmetros...');
        Writeln('   (se deseja modificar alguma coisa referente à
            execuçÃo do programa!)');
        Writeln;
        Writeln('3) Mostre-me uma animaçÃo de um período!');
        Writeln('   (imagem muito menos perfeita, constituída de poucos
            triângulos apenas para cálculo rápido)');
        Writeln;
        Writeln('4) Renderize uma imagem pixel-a-pixel da onda.');
        Writeln('   (imagem impecável em qualidade. Mais demorada também,
            de longe!)');
        Writeln('   (este modo evita qualquer defeito na imagem que possa
            aparecer)');
        Writeln;
        Writeln('5) Sair do Programa');
        Writeln;
        Write('Sua Escolha: ');
        Readln(choice);
End;

{mais um último procedimento para sintetizar toda a versão anterior do
programa....}

procedure OperationDraw;
begin
     gD:=Detect;gM:=GetMaxMode;
     Initgraph(gD,gM,'');
```

```
     If GraphResult<>grOK then Halt(1);
     maxx:=GetMaxX;
     maxy:=GetMaxY;
     For ccon1:=1 to 255 do
     SetRGBPalette(ccon1,ccon1,ccon1,ccon1);
     Moveto(10,10);
     Outtext('Interfer^ncia entre duas ondas/fun‡äes de onda, V1.00');
     Moveto(10,30);
     Outtext('Por David Dobrigkeit Chinellato');
     Moveto(10,50);
     Outtext('Calculando figura com 131072 faces...');
           camplace;
           locatepoints;
           suratt3d;
           morphingtime;
           suratt;
           drawnow;
           drawplaneofintersect;
           drawGraph;
           Readln;
     CloseGraph;
End;

procedure RespondToUserNow;
begin
        If choice=1 then OperationDraw;
        If choice=2 then ChangeParams;
        If choice=3 then Anime;
        If choice=5 then XExit;
        If choice=4 then Scanlinetodeath;
End;

{========================================================================}

{=5.O Programa, De fato!=================================================}

begin
     ClrScr;
```

```
    OpenParams;
    xw:=-xc;yw:=-yc;zw:=-zc;
    Repeat
         DrawMenu;
         RespondtoUserNow;
    until choice=5;
    If choice=5 then XExit;
End
```

Apêndice B: *Extrato de Introdução à Interferência, reilustrado*

Será aqui reproduzido um trecho de um livro introdutório de física do nível secundário fazendo uso de figuras que o nosso programa concebeu. Ao longo do texto, a utilidade das imagens por nós geradas se tornará bastante óbvia. O livro escolhido para a reprodução foi A.Máximo e B.Alvarenga, "Física – Volume Único", editora Scipione, páginas 642-644. O livro é tipicamente introdutório à física e é de nível secundário.

12.8 Interferência

Figura de Interferência

Na seção 12.5, a **interferência** de duas ondas foi mencionada como sendo um fenômeno típico do movimento ondulatório, e o fato de se observar esse fenômeno também com a luz conduziu à aceitação, praticamente definitiva, de que ela é um tipo especial de onda.

Figura 12-119a: Padrão de interferência.

Para você entender o que é o fenômeno de interferência, observe a figura 12.119a, na qual temos duas pequenas esferas, acionadas por um dispositivo especial, batendo **simultaneamente** na superfície da água de um tanque e gerando duas ondas circulares que se propagam na superfície do líquido.[2] Essas esferas são, portanto, **fontes de ondas**, que vamos designar de F_1 e F_2.

As duas ondas originadas em F_1 e F_2 evidentemente irão se sobrepor enquanto se propagam, e a figura 12-119a representa exatamente o **resultado dessa superposição**.

---

[2] Esta situação é apenas um exemplo; nosso programa pode representar isto – ou pelo menos o caso ideal disto – perfeitamente.

Dizemos que as ondas se **inteferiram** e a configuração adquirida pela superfície do líquido, mostrada na figura, é denominada figura de interferência.

Podemos observar que, na figura de interferência, existem várias retas que passam pelo ponto médio entre as fontes e, entre estas linhas, temos cristas e vales se propagando, afastando-se das fontes, como evidenciado na figura 12-119b.

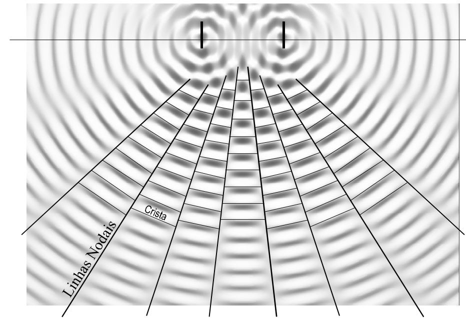

Figura 12-119b: Linhas nodais e propagação de cristas e vales duplos.

Por que se forma a figura de interferência

A formação da figura de interferência deve-se basicamente aos dois fatos seguintes:

- As linhas mencionadas são constituídas por pontos que estão **permanentemente em repouso**, apesar de estarem sendo atingidos simultaneamente por duas ondas. Ocorre que as ondas chegam a cada um destes pontos de tal modo que a crista de uma delas coincide com o vale da outra e, por isso, os deslocamentos que cada uma iria produzir **se anulam**. Essa situação é descrita dizendo-se que houve **interferência destrutiva** das ondas, o ponto em repouso é denominado **nó** e cada linha constituída de nós é uma **linha nodal**.
- Entre duas linhas nodais, a crista de uma onda chega juntamente com a crista de outra onda, o mesmo ocorrendo com os vales dessas ondas. Então, nesses pontos, os deslocamentos que cada uma provocaria individualmente se adicionam, gerando **duplas cristas** e **duplos vales** que se propagam entre as linhas nodais (figura 12-119). Portanto, entre as linhas nodais temos uma **interferência construtiva** das duas ondas, isto é, um ponto nesta posição oscila com uma amplitude igual à soma das amplitudes das ondas que se interferiram.

Em resumo:

Em uma figura de interferência observam-se linhas nodais, constituídas por pontos permanentemente em repouso (interferência destrutiva), e duplas cristas e duplos vales sucessivos (interferência construtiva), propagando-se entre as linhas nodais.

Interferência com a luz

Devemos ao cientista inglês T. Young a primeira comprovação experimental de que era possível ocorrer o fenômeno de interferência com dois feixes luminosos. Na figura 12-120 vemos, esquematicamente, uma montagem semelhante à usada por ele. Temos, nessa montagem:

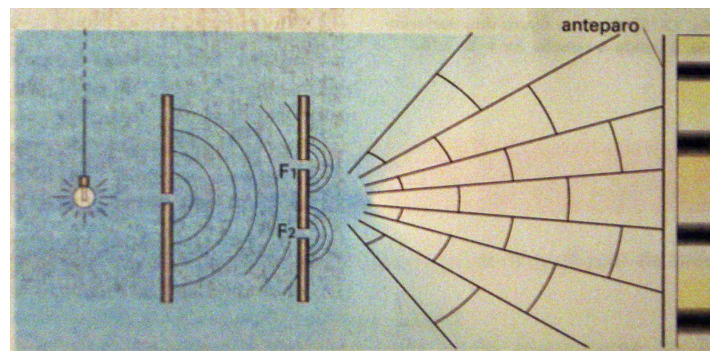

Figura 12-120: Montagem semelhante à de Young para interferência entre duas ondas luminosas.

- Uma lâmpada emitindo luz que se difrata ao passar pelo pequeno orifício O;
- A onda luminosa difratada em O dirige-se para os dois pequenos orifícios F_1 e F_2 eqüidistantes de O;
- A luz torna a se difratar em F_1 e F_2 e, assim, esses orifícios se comportam como duas fontes luminosas, semelhantes às fontes da figura 12-119;
- As ondas luminosas então se sobrepõem, dando origem a uma figura de interferência, com a formação de linhas nodais e com duplas cristas e duplos vales propagando-se entre elas (figura 12-120);

- A verificação de que realmente ocorreu a interferência pode ser feita colocando-se um anteparo, ou um filme fotográfico, na posição indicada na figura 12-120. Obtém-se o resultado mostrado na figura 12-121, onde as faixas escuras correspondem às regiões em que houve interferência destrutiva das ondas luminosas (regiões nodais) e as faixas claras são as regiões do anteparo (ou filme) atingidas por duplas cristas e duplos vales da onda de luz (essas faixas claras e escuras costumam ser denominadas **franjas de interferência**). Esta constitui uma evidência experimental extremamente favorável à teoria ondulatória da luz.

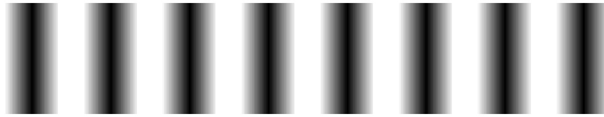

Figura 12-121: Padrão de interferência formado em anteparo.

- Uma forma mais precisa de enxergar o sistema seria pensar exatamente na sobreposição das duas ondas. Neste caso, basta contemplar a figura 12-122 e será possível visualizar o que o anteparo recebe em intensidade das duas ondas.

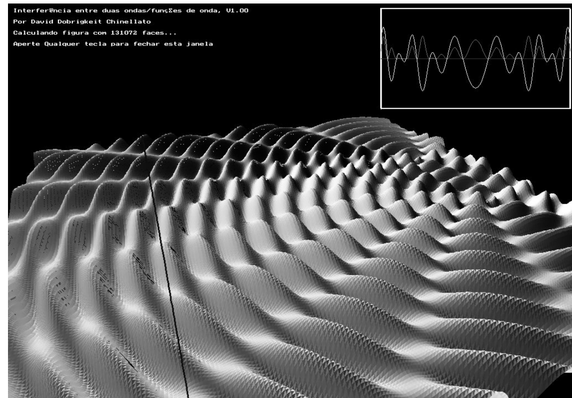

Figura 12-122: Interferência entre as duas ondas da figura 12-119a vista em detalhe.