# Um Ambiente Distribuído de Visualização com Suporte para Geometria Projetiva Orientada*

PEDRO J. DE REZENDE[1]    CÉSAR N. GON[1]

[1]Instituto de Computação
Universidade Estadual de Campinas
13081-970 Campinas, SP, Brasil
{rezende|gon}@dcc.unicamp.br

**Abstract.** This paper describes the design of GeoPrO: a distributed programming environment for geometric visualization. We present an overview of its classes that allow for easy extensibility and portability. Due to a client-server architecture, comprised of a kernel, with multiple contexts, applications and visualizers, GeoPrO supports distributed execution over a heterogeneous network. Visualizers are currently available for the planar and the spheric models of the oriented projective geometry, running on SGI workstations, while another is being implemented in Java for multi-platform support.

## 1 Introdução

GeoPrO é um ambiente de software para implementação de algoritmos geométricos. O ambiente GeoPrO foi projetado como uma ferramenta de auxílio ao estudo, desenvolvimento e depuração de aplicações na área de geometria computacional. Além disso, o ambiente foi estruturado de modo a permitir a sua utilização em projetos de outras áreas que realizam algum tipo de computação geométrica e onde podem ser de grande valia recursos de visualização geométrica (p.ex., programação linear, robótica, GIS, etc.).

A arquitetura aberta do ambiente é o primeiro passo no sentido de permitir sua evolução através de contribuições dos próprios usuários do sistema. Para isso, serão criados mecanismos de comunicação via Internet de modo a permitir uma maior interação entre estes usuários, incluindo um repositório (acessível via WWW) para tais contribuições e para distribuição de novas versões do GeoPrO.

## 2 Trabalhos relacionados

**GeoLab** [5, 1]. Foi desenvolvido como um ambiente de programação para implementação, teste e animação de algoritmos geométricos. GeoLab utiliza bibliotecas compartilhadas de algoritmos e uma abordagem incremental para a composição de novos tipos de objetos geométricos e módulos funcionais externos acessados via ligação dinâmica. Foi escrito em C++ e roda sobre a plataforma SunOS/XView.

*Pesquisa desenvolvida com suporte financeiro parcial de: Capes, CNPq, ProTeM-CC — projeto GEOTEC, e Fapesp.

**LEDA** [7]. É uma biblioteca de tipos de dados e algoritmos implementada segundo uma abordagem orientada a objetos através de um conjunto de classes C++. Uma pequena parte da biblioteca é dedicada a geometria computacional, com algoritmos para resolução de problemas geométricos no plano euclidiano. LEDA está disponível para as mais diversas plataformas sendo considerada um modelo para o desenvolvimento de bibliotecas de software.

**GeoSheet** [6]. É uma ferramenta para visualização de algoritmos geométricos em ambientes distribuídos. GeoSheet provê visualização interativa de programas para depuração através de uma interface para entrada e saída de objetos geométricos baseada no programa Xfig. Uma aplicação pode utilizar diversas "planilhas" para realizar a visualização remota de objetos geométricos. A implementação foi realizada em C++.

**XYZ GeoBench** [9, 10]. Foi projetado como um ambiente de programação geométrica, provendo ferramentas para criação, edição, e manipulação de objetos geométricos. A ênfase é na robustez de implementação de algoritmos fundamentais. É composto de uma interface gráfica e uma biblioteca de algoritmos, sendo implementado em *Object Pascal* para Macintosh.

**GeomView** [8]. É uma interface para visualização e manipulação de objetos geométricos, podendo ser utilizado como um visualizador para objetos estáticos ou para objetos dinamicamente produzidos por outros programas. GeomView roda sobre estações IRIS

Silicon Graphics e estações NeXT.

## 3 Por que um ambiente de visualização geométrica?

Tradicionalmente, algoritmos geométricos manipulam objetos dotados de descrições e funcionalidades geométricas. Por exemplo, um objeto que representa uma subdivisão planar pode ter associado a ele, além de sua descrição geométrica (vértices, faces e arestas), métodos que implementam funcionalidades como a localização de pontos, percursos, etc.

Por outro lado, é raro encontrarmos métodos para visualização ou entrada destes objetos através de uma interface gráfica. A implementação *ad-hoc* dessas funcionalidades é normalmente tediosa e específica para cada plataforma, o que obriga o implementador a desviar sua atenção para detalhes relacionados ao dispositivo gráfico, à definição de uma interface com o usuário, etc.

A utilização de programas dedicados à visualização geométrica [1, 10, 8] também apresenta uma série de limitações, como a dificuldade de integração entre a aplicação do usuário e o programa de visualização. Além disso, estes programas normalmente estão disponíveis para apenas uma plataforma e/ou linguagem de programação e os recursos de visualização e entrada de objetos não são facilmente estendíveis para incorporar características que atendam às necessidades específicas das aplicações.

O ambiente GeoPrO foi desenvolvido como um ambiente aberto tendo como meta os seguintes requisitos:

- *Facilidade de integração entre as aplicações e o ambiente.* A utilização de uma abordagem cliente-servidor, possibilita que, em um ambiente de rede[1], *qualquer aplicação* possa facilmente utilizar os recursos de visualização do GeoPrO.

- *Reaproveitamento de código.* Isto é conseguido através de uma metodologia, baseada no paradigma da programação orientada a objetos, de integração entre classes geométricas existentes e o ambiente GeoPrO, descrita na seção 4.3.2.

- *Abstração de dados.* A interação entre os objetos da aplicação e o ambiente é feita através da utilização de um protocolo de descrição geométrica (descrito na seção 4.1) que deixa transparente ao programador detalhes não relacionados ao algoritmo em questão.

[1]A versão atual do ambiente é baseada no protocolo TCP/IP.

- *Extensibilidade.* O ambiente foi concebido para ser estendido de acordo com as necessidades do usuário. A idéia é prover um conjunto mínimo de ferramentas para que o usuário-programador derive soluções de visualização e de entrada de dados cada vez mais sofisticadas.

## 4 A estrutura básica do ambiente GeoPrO

Em um alto nível de abstração, o ambiente consiste de três partes básicas: o *núcleo* do GeoPrO, os *visualizadores geométricos* e as *aplicações* do usuário. Uma aplicação pode ser a implementação de um algoritmo geométrico ou um módulo de um sistema maior para o qual se deseje algum tipo de visualização geométrica (veja figura 1).

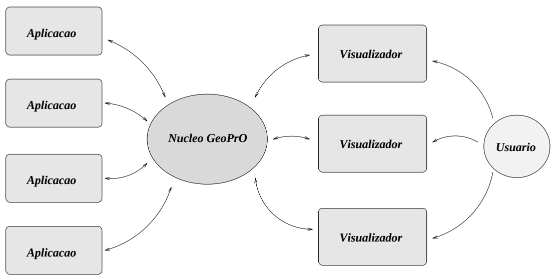


Figura 1: Componentes do ambiente GeoPrO

Quando uma aplicação deseja usar as facilidades do ambiente, ela conecta-se ao núcleo do GeoPrO, tornando-se um novo *cliente*. Então, esta aplicação passa a ter à disposição uma série de recursos para realizar a visualização de objetos geométricos ou para obter entradas de dados específicas do usuário. Através desses recursos, o algoritmo pode enviar requisições para um conjunto de visualizadores que são responsáveis pelo desenho dos objetos e por oferecer uma interface gráfica amigável para que o usuário possa manipulá-los ou atender a um pedido de entrada de dados. O usuário pode optar por usar um visualizador já incluído no ambiente GeoPrO ou por implementar um novo visualizador especialmente projetado para a sua aplicação. Detalhes sobre o funcionamento das diversas partes do ambiente e a integração entre elas serão vistos em seções adiante.

**Operação em rede** GeoPrO foi projetado para trabalhar *eficientemente* em um ambiente de rede. O formato de transmissão dos comandos das aplicações para o núcleo e deste para os visualizadores é definido de modo a ser independente da plataforma de *hardware*. Assim, o núcleo, as aplicações e os visualizadores podem estar rodando distribuídos em máquinas de uma rede heterogênea. Por exemplo, o usuário pode rodar os seus algoritmos em uma máquina paralela e ver o resultado geométrico ser mostrado em um visualizador rodando em uma estação Silicon Graphics, com o núcleo sendo executado em uma terceira plataforma.

As seções seguintes abordam a estrutura básica do ambiente GeoPrO.

### 4.1 Objetos geométricos primitivos

Existe uma grande gama de algoritmos geométricos para o plano que lidam com objetos razoavelmente complexos como triangularizações, diagramas de Voronoi, árvores espalhadas, etc. Contudo, estes objetos podem, em geral, ser construídos a partir de um pequeno número de objetos geométricos primitivos.

GeoPrO prevê um conjunto básico de objetos geométricos, e todas as operações de desenho e de entrada de dados são feitas através desse conjunto de objetos. Desse modo, qualquer objeto descrito em termos desses objetos primitivos pode interagir com o ambiente. Assim, o usuário é encorajado a escrever as suas próprias bibliotecas de objetos mais complexos, descrevendo-os a partir dos objetos básicos do GeoPrO.

Os objetos primitivos são:

- *pontos* - uma lista de um ou mais pontos,
- *linhas* - uma lista de uma ou mais retas,
- *segmentos* - uma lista de um ou mais pares de vértices interpretados como segmentos de retas,
- *círculos* - uma lista de um ou mais pares de vértices interpretados como centro e ponto na borda de um círculo,
- *linha poligonal* - uma seqüência de vértices descrevendo uma linha poligonal (possivelmente aberta),
- *polígono* - uma seqüência de vértices descrevendo um polígono.

Esses objetos básicos são hierarquizados da forma descrita na figura 2. Maiores detalhes sobre a implementação desses objetos podem ser obtidos em [4].

A partir da superclasse *NetObj*, são derivadas três subclasses:

- *NetSingle:* classe abstrata que representa os objetos descritos por uma seqüência de vértices, ou seja, *NetPoints*, *NetLines*, *NetPolygonalLine* e *NetPolygon*. Cada vértice é representado por uma instância da classe *Coord*, ou seja, uma coordenada homogênea com sinal.

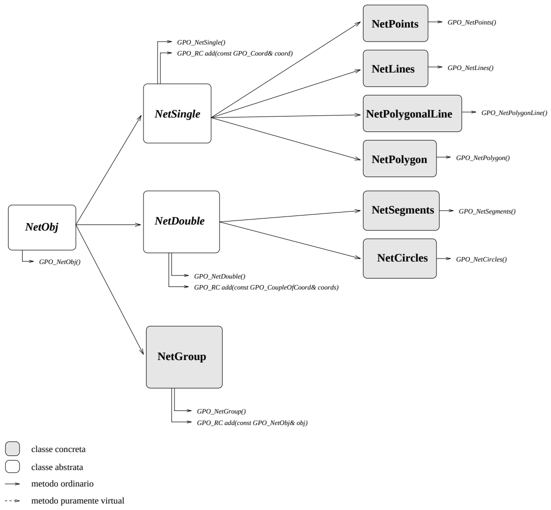


Figura 2: Hierarquia dos objetos geométricos primitivos

- *NetDouble:* classe abstrata que representa os objetos descritos por uma seqüência de pares de vértices, ou seja, *NetSegments* e *NetCircles*. Cada par de vértices é representado por uma instância da classe *CoupleOfCoords*, ou seja, um par de coordenadas homogêneas com sinal.
- *NetGroup:* representa a entidade de agrupamento de objetos. Uma instância dessa classe descreve uma seqüência de instâncias da superclasse *NetObj* ou de suas classes derivadas.

Esse conjunto de objetos básicos define o que chamaremos de *protocolo de descrição geométrica* (PDG) e toda descrição geométrica dentro do ambiente GeoPrO é feita através deste protocolo.

O modelo geométrico utilizado é a geometria projetiva orientada [11] através da descrição dos objetos geométricos primitivos por coordenadas homogêneas com sinal.

Como em [3] descrevemos uma biblioteca de primitivas geométricas para a geometria projetiva orientada que serve de base para a implementação de algoritmos geométricos com robustez garantida, omitiremos detalhes desta biblioteca aqui. Ressaltamos, porém, que ela é um elo importante na integração consistente e harmoniosa entre geometria projetiva orientada, perturbação simbólica para tratamento de casos degenerados e aritmética exata. Referenciamos o leitor interessado ao trabalho [3] publicado nos anais deste Simpósio em 95.

### 4.2 O Núcleo do GeoPrO

Uma aplicação do usuário pode enviar *comandos* para os visualizadores, que podem ser ações ou requisições. As primeiras determinam mudanças nos objetos mostrados e as últimas representam pedidos de novos objetos.

O *núcleo* do GeoPrO pode ser visto como uma interface entre a aplicação do usuário e os visualizadores. O principal objetivo do núcleo é rotear os comandos da aplicação para os visualizadores. Além disso, ele também é responsável por manter toda informação sobre o conjunto de objetos geométricos já criados (e não removidos) pelas aplicações. Esses objetos são agrupados de maneira disjunta em *contextos*. Cada aplicação está associada a um único contexto do núcleo, que compreende o escopo dos objetos "visíveis" para aquela aplicação. Um visualizador também tem o seu escopo restrito a um único contexto.

A idéia de múltiplos contextos é permitir que diversos grupos de aplicações e visualizadores compartilhem os recursos de um único núcleo sem que haja nenhum tipo de interferência entre eles. Para isso, basta que estes grupos se associem a diferentes contextos. O suporte a múltiplos contextos é discutido na seção 5.

### 4.2.1 O funcionamento do núcleo

Esta seção apresenta uma visão geral do modo de funcionamento do núcleo. Detalhes sobre a implementação das classes que compõem o núcleo podem ser obtidos em [4].

Internamente, o núcleo do GeoPrO pode ser visto como uma máquina de estados que responde a eventos de oito tipos distintos.

Os eventos gerados pelas aplicações são:

- *Cadastramento de aplicações*
- *Inserção de objetos geométricos*
- *Remoção de objetos geométricos*
- *Requisição de objetos geométricos*
- *Descadastramento de aplicações*

Os eventos gerados pelos visualizadores são:

- *Cadastramento de visualizadores*
- *Resposta à requisição de objetos geométricos*
- *Descadastramento de visualizadores*

Por limitação de espaço, omitimos aqui maiores detalhes sobre o funcionamento do núcleo, mas estes podem ser encontrados em [4].

## 4.3 Aplicações do usuário

Uma vez conectadas ao núcleo do GeoPrO, as aplicações do usuário estão habilitadas a inserir objetos geométricos nos contextos do núcleo para que sejam mostrados pelos visualizadores ativos ou remover objetos destes contextos para que sejam apagados dos visualizadores. Finalmente, as aplicações podem requisitar alguma entrada específica ao usuário que pode escolher o visualizador mais conveniente para prover os objetos requisitados.

### 4.3.1 A classe *Application*

Nesta seção, apresentamos a interface entre a aplicação e o núcleo no nível da linguagem de programação. Essa interface está disponível para a linguagem C++ e, em breve, estará disponível para as linguagens Modula-3 e Java.

O objetivo desta seção é apresentar a interface pública da classe *Application* (veja figura 3). Detalhes sobre a implementação desta classe podem ser vistos em [4].

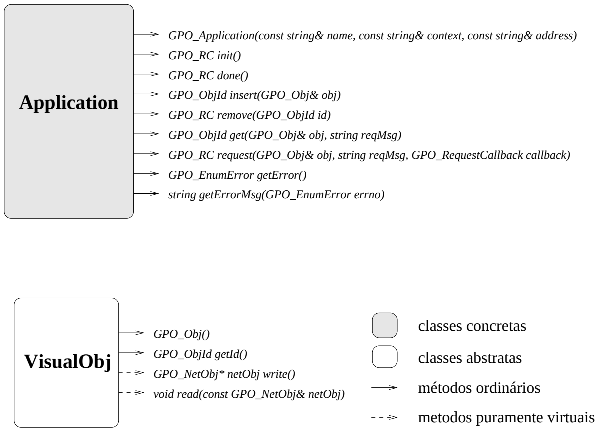


Figura 3: As interfaces públicas das classes *Application* e *VisualObj*

A interface entre uma aplicação e o ambiente GeoPrO é feita através da classe *Application*. Uma instância dessa classe representa uma aplicação e está sempre relacionada a um determinado contexto de um núcleo.

A classe *Application* utiliza a classe genérica *VisualObj*, que representa a entidade de integração entre os objetos geométricos da aplicação e o ambiente GeoPrO. Qualquer instância de uma subclasse da classe *VisualObj* poderá, por alomorfismo, representar um objeto no ambiente GeoPrO. A classe *VisualObj* e a metodologia de integração entre os objetos da aplicação e o ambiente são descritas na seção 4.3.2.

Ao criar uma instância da classe *Application* o usuário define o núcleo e o contexto ao qual ele deseja se conectar. A definição do núcleo é feita através do endereço *Internet* de uma máquina onde o núcleo do GeoPrO está rodando. Se este parâmetro for omitido, é assumido que o núcleo é local. O contexto também pode ser omitido e neste caso é assumido um contexto especial denominado "*default*". Usando a interface C++, a criação de uma instância é feita através da utilização do seguinte construtor:

```
GPO_Application(const string& name,
    const string& context = DEFAULT_CONTEXT,
    const string& address = DEFAULT_KERNEL_ADDRESS)
```

Uma vez criada a instância da classe, a aplicação deve contectar-se ao núcleo através do método *init*. Quando este método é evocado, a aplicação registra-se como um novo cliente junto ao núcleo do GeoPrO, podendo então usar os recursos do ambiente. Através da interface C++, essa iniciação é feita utilizando-se o seguinte método da classe *Application*:

```
GPO_RC init()
```

Para finalizar as suas operações com o núcleo do GeoPrO, a aplicação utiliza o método *done*, disponível na classe *Application*. Na interface C++ temos:

```
GPO_RC done()
```

Para inserir um objeto no contexto corrente do núcleo, a aplicação deve utilizar o método *insert*. Quando um objeto é adicionado a um contexto do núcleo, todos os visualizadores registrados àquele contexto receberão o novo objeto. Esse novo objeto, que agora faz parte de um contexto do núcleo, passa a ser *persistente*, recebendo um identificador (*id*) que servirá para identificá-lo univocamente perante o núcleo, as aplicações e os visualizadores. Em C++ temos:

```
GPO_ObjId insert(GPO_VisualObj& obj)
```

Para remover um objeto do núcleo, e conseqüentemente de todos os visualizadores, basta utilizar o método *remove*, passando como argumento o *id* do objeto a ser removido do contexto corrente do núcleo. Na interface C++ temos:

```
GPO_RC remove(GPO_ObjId id)
```

Quando uma aplicação deseja receber um objeto como entrada, ela pode enviar dois tipos de requisição para o núcleo: síncrona ou assíncrona. Na primeira, a aplicação ficará bloqueada até que a entrada desejada seja provida por algum visualizador conectado ao núcleo. Na segunda, a aplicação não ficará bloqueada e receberá o objeto requisitado de forma assíncrona através da evocação de uma *callback*. Na interface C++, estes serviços estão disponíveis através dos seguintes métodos:

```
GPO_ObjId get(GPO_VisualObj& obj,
              const string reqMsg)
GPO_RC request(GPO_VisualObj& obj,
              const string reqMsg,
              GPO_RequestCallback callback)
```

Onde o tipo *GPO_RequestCallback* é definido como:

```
typedef
  void (GPO_RequestCallback)(GPO_VisualObj& obj)
```

A verificação de erro é feita através dos métodos *getError* e *getErrorMsg*. O primeiro contém sempre um código de erro referente à última chamada à interface da classe *Application*. O segundo retorna uma mensagem descritiva para um determinado erro. Na interface C++, estes serviços estão disponíveis através dos seguintes métodos:

```
GPO_EnumError getError()
string getErrorMsg(GPO_EnumError errno)
```

### 4.3.2 Visualização de algoritmos usando GeoPrO

Esta seção tem por objetivo descrever a metodologia de utilização do ambiente GeoPrO para visualização e requisição de objetos geométricos.

Uma *classe geométrica* representa um específico objeto geométrico dotado de um conjunto de estruturas de dados e funcionalidades que são utilizadas na construção de algoritmos geométricos. Uma *classe geométrica visual* é uma extensão de uma classe geométrica que contém uma componente funcional responsável pela integração entre esta classe geométrica e o ambiente GeoPrO.

Uma classe geométrica visual é construída por derivação múltipla a partir de uma classe geométrica e da classe genérica *VisualObj*, conforme ilustra a figura 4. A classe resultante herda todas características geométricas da classe original além da especificação funcional dos dois métodos de visualização oriundos da superclasse *VisualObj*: *read* e *write*. Em última análise, a implementação destes métodos segundo o protocolo de descrição geométrica (PDG) (descrito na seção 4.1) constitui o trabalho de integração entre uma classe geométrica qualquer e o ambiente GeoPrO.

Maiores detalhes sobre a implementação dos métodos de visualização são encontrados em [4].

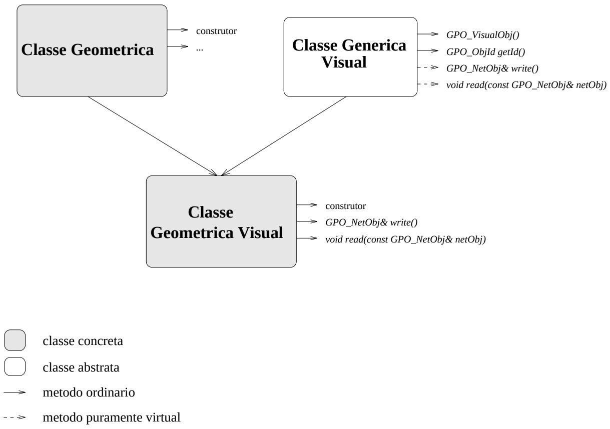


Figura 4: Construção de uma classe geométrica visual

### 4.3.3 A biblioteca *Visual-LEDA*

Como exemplo da utilização do ambiente GeoPrO, realizamos a implementação da biblioteca *Visual-LEDA*, uma extensão da coleção de tipos de dados e algoritmos geométricos disponíveis na biblioteca LEDA para que possam ser visualizados através do ambiente GeoPrO.

Essa extensão foi realizada segundo a metodologia de integração descrita na seção 4.3.2, mostrando a sua eficiência na integração entre algoritmos e objetos geométricos existentes e o ambiente GeoPrO. Utilizando os tipos estendidos, diversos algoritmos geométricos, disponíveis na biblioteca LEDA, foram facilmente integrados ao ambiente.

## 4.4 Visualizadores geométricos

Os *visualizadores geométricos* prevêem um modo fácil de permitir a interação entre os usuários e as aplicações. Um visualizador é um processo que, uma vez conectado ao núcleo, é capaz de desenhar os objetos geométricos primitivos de acordo com um específico modelo geométrico. Ele também provê uma interface gráfica para entrada de objetos como resposta a requisições dos algoritmos (ou seja, das aplicações clientes conectadas ao núcleo). De maneira análoga às aplicações, um visualizador está sempre associado a um único contexto do núcleo.

Em adição a alguns visualizadores que o usuário tem à disposição (já implementados no ambiente GeoPrO), ele pode implementar um novo especialmente projetado para a sua aplicação. Tal implementação é facilitada se for feita através da derivação de um novo visualizador a partir de um pré-existente, segundo uma abordagem de programação orientada a objetos.

A interface entre um visualizador e o núcleo é feita através da classe *Visualizer*. Todos os visualizadores do ambiente GeoPrO são instâncias de subclasses da classe *Visualizer*. Veja figura 5. A metodologia de implementação de novos visualizadores para o ambiente através da classe *Visualizer* é apresentada em [4]. Essa interface está disponível para a linguagem C++ e, em breve, estará disponível para a linguagens Modula-3 e Java.

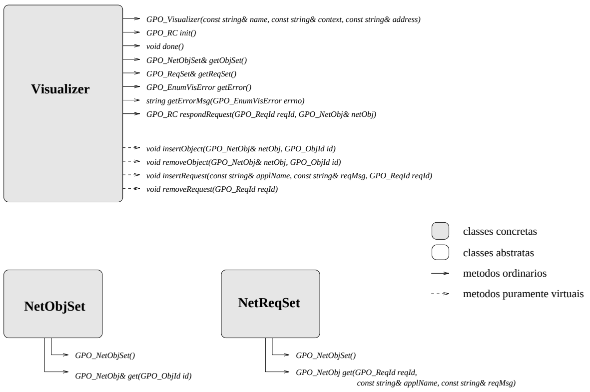


Figura 5: A classe *Visualizer*

### 4.4.1 Visualizadores implementados

O modelo esférico do plano projetivo orientado $\mathbf{T}^2$ [11, 2], facilita a visualização da sua topologia e de suas propriedades geométricas, principalmente em relação aos pontos no infinito. Além disso, é uma ferramenta visual para auxiliar na interpretação de problemas e na derivação de algoritmos.

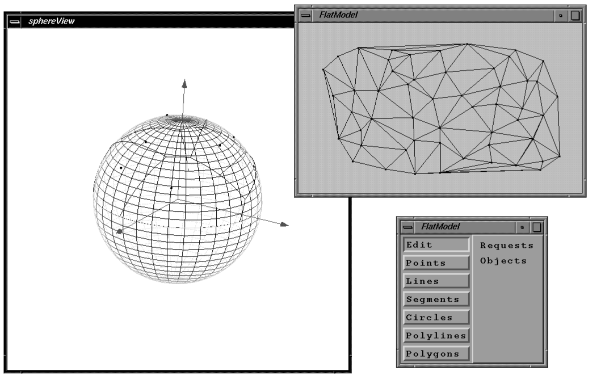


Figura 6: Visualizadores para o ambiente GeoPrO

Um visualizador para este modelo foi implementado, possuindo uma interface gráfica que permite a visualização dos objetos básicos tratados pelos algoritmos geométricos conectados ao núcleo do ambiente GeoPrO. A implementação foi realizada sobre a plataforma IRIS Silicon Graphics, podendo ser portada para qualquer plataforma que suporte *OpenGL* e *X Window*.

Além do visualizador esférico, dois visualizadores planares foram implementados, um sobre *X* e outro sobre *Motif+OpenGL* (veja figura 6). Maiores detalhes sobre estes visualizadores podem ser encontrados em [4].

Finalmente, está em andamento o desenvolvimento de um visualizador planar em Java que permitirá a visualização de algoritmos via *Web*, o que deverá ser de grande valia para a comunidade científica na divulgação de trabalhos relacionados ao desenvolvimento de algoritmos geométricos.

## 5 Características de projeto do GeoPrO

### Suporte a execução distribuída

O ambiente GeoPrO suporta múltiplas aplicações e múltiplos visualizadores executando simultaneamente em máquinas (homogêneas ou heterogêneas) distribuídas. Esse tipo de operação será referenciado como *execução distribuída*.

O suporte a execução distribuída tem como requisito básico a solução do seguinte problema: como as aplicações e os visualizadores identificam o núcleo ao qual desejam conectar-se?

Como a comunicação entre os processos é baseada no protocolo TCP/IP, a solução adotada foi a padronização de duas portas TCP/IP *Internet*, uma para requisição de conexão de aplicações e outra de visualizadores. Deste modo, um cliente (aplicação ou visualizador) identifica o núcleo ao qual irá se conectar apenas especificando o endereço da máquina onde este está sendo executado. Este esquema limita a execução de no máximo um núcleo por máquina, o que não é restritivo devido ao suporte a múltiplos contextos, descrito na seção 5.

As classes que implementam o suporte de comunicação à execução distribuída estão implementadas na biblioteca GeoPrO IPC, descrita em [4].

### Persistência dos objetos de visualização

Quando um objeto é inserido em um contexto do núcleo ele se torna *persistente*, recebendo um identificador único em relação a todos os objetos de todos os contextos do núcleo. Esta unicidade permitirá a realização de uniões de contextos sem conflitos de identificadores.

A criação de um objeto persistente pode ocorrer por meio de dois eventos distintos:

- a inserção de um objeto em um contexto do núcleo feita por uma aplicação, ou,
- a resposta de um visualizador a uma requisição de entrada objeto feita por alguma aplicação.

Um objeto persistente está somente associado ao contexto ao qual ele foi adicionado e não à aplicação que realizou a inserção ou ao visualizador que respondeu a uma requisição.

Mesmo que o programa (aplicação ou visualizador) que gerou o evento que inseriu o objeto no núcleo termine, este permanecerá no ambiente, sendo apresentado nos visualizadores e disponível para servir de matriz para objetos enviados em resposta a pedidos de entrada feitos pelas aplicações. Este objeto existirá até que alguma aplicação o remova do contexto do núcleo ou até um eventual encerramento das operações do núcleo.

Uma extensão natural das funcionalidades do núcleo GeoPrO será a capacidade de armazenamento e recuperação dos contextos em uma base de dados. Isso expandirá a capacidade de armazenamento de informação do núcleo e permitirá que a existência dos contextos não fique restrita ao período de execução do núcleo.

Persistência de objetos é um requerimento fundamental para que, em um ambiente de programação distribuída, as diversas aplicações possam compartilhar resultados.

### Suporte a múltiplos contextos

Um contexto no núcleo tem associado a ele um grupo de aplicações, um grupo de visualizadores, um conjunto de objetos geométricos e um conjunto de requisições de objetos. Cada aplicação, visualizador, objeto ou requisição está associada a um único contexto. O suporte a *múltiplos contextos* permite que diversos grupos de aplicações e visualizadores estejam associados a contextos distintos, manipulando somente conjuntos de objetos e requisições disponíveis no contexto a que estão associados.

Múltiplos contextos são uma característica fundamental no GeoPrO para permitir que diversos usuários compartilhem os recursos do ambiente de maneira independente.

### Extensibilidade e escalabilidade

A abordagem orientada a objetos utilizada no projeto dos componentes do GeoPrO, a arquitetura multi-plataforma e o suporte a execução distribuída permitem que o ambiente seja estendido de acordo com as necessidades do usuário. Objetos cada vez mais complexos e visualizadores dotados de novos e sofisticados recursos de visualização e entrada de dados podem ser facilmente integrados ao ambiente.

Além disso, a flexibilidade de utilização de visualizadores para diferentes plataformas, independentemente das plataformas que hospedam as aplicações, permite que o usuário dimensione a utilização dos recursos computacionais disponíveis de acordo com as características de suas aplicações. Assim, se um *terminal X* deixa de ser suficiente para a demanda de visualização de uma determinada aplicação, o usuário poderá optar por utilizar um visualizador em uma plataforma com recursos gráficos mais sofisticados (p.ex., uma estação Silicon Graphics) que lhe esteja disponível.

## 6 Implementação

O pacote básico do ambiente GeoPrO compreende:

- o núcleo do GeoPrO (apresentado na seção 4.2),
- a interface *Application* para integração de aplicações ao ambiente (apresentada na seção 4.3) e
- a interface *Visualizer* para construção de novos visualizadores (apresentada na seção 4.4).

O pacote básico do ambiente GeoPrO é implementado por uma família de classes C++, podendo ser utilizada praticamente com qualquer compilador C++ (por exemplo, GNU G++, Sun C++, AIX CSET++ e HP C++).

Na versão atual, toda a comunicação entre processos é feita através do protocolo TCP/IP, o que estabelece uma dependência seguramente não restritiva devido a elevada disponibilidade deste protocolo nas diferentes plataformas de *hardware/software*.

A compilação do ambiente ainda depende da disponibilidade da biblioteca LEDA [7], também altamente disponível para as mais diversas plataformas.

## 7 Conclusões

GeoPrO é um ambiente distribuído para visualização de algoritmos geométricos. A utilização de uma abordagem cliente-servidor facilita a utilização do ambiente pelas aplicações, independentemente da linguagem de programação ou da plataforma de origem das mesmas. A metodologia de integração entre classes geométricas existentes e o ambiente GeoPrO permite um alto nível de abstração e facilita a reutilização de código, um requisito fundamental para um ambiente de visualização geométrica. Por fim, o ambiente GeoPrO foi estruturado para ser estendido de acordo com as necessidades do usuário, permitindo que este derive soluções de visualização e entrada de dados cada vez mais sofisticadas.

## Referências


[1] P. J. de Rezende and W. R. Jacometti. GeoLab: An environment for development of algorithms in computational geometry. In *Proc. 5th Canad. Conf. Comput. Geom.*, pages 175–180, Waterloo, Canada, 1993.

[2] P. J. de Rezende and J. Stolfi. *Fundamentos de Geometria Computacional.* IX Escola de Computação, 1994.

[3] P. J. de Rezende e C. N. Gon. GeoPrO: Geometria projetiva orientada com tratamento de degenerações. In *VII Simpósio Brasileiro de Computação Gráfica e Processamento de Imagens*, pages 315–316, 1995.

[4] C. N. Gon. Computação exata em geometria projetiva orientada e tratamento de degenerações. Master's thesis, IC - UNICAMP, 1996.

[5] W. R. Jacometti. GeoLab – um ambiente para desenvolvimento de algoritmos em geometria computacional. Master's thesis, DCC - IMECC - UNICAMP, 1992.

[6] D. T. Lee, S.-M. Sheu, and C.-F. Shen. Geosheet: A distributed visualization tool for geometric algorithms. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, 1995.

[7] K. Mehlhorn and S. Näher. LEDA, a library of efficient data types and algorithms. Report A 04/89, Fachber. Inform., Univ. Saarlandes, Saarbrücken, West Germany, 1989.

[8] T. Munzner, S. Levy, and M. Philips. Geomview: A system for geometric visualization. In *Proc. 11th Annu. ACM Sympos. Comput. Geom.*, pages C12–C13, 1995.

[9] P. Schorn. An object-oriented workbench for experimental geometric computation. In *Proc. 2nd Canad. Conf. Comput. Geom.*, pages 172–175, 1990.

[10] P. Schorn. Implementing the XYZ GeoBench: A programming environment for geometric algorithms. In *Proc. Internat. Workshop Comput. Geom. CG '91*, volume 553 of *LNCS*, pages 187–202. 1991.

[11] J. Stolfi. *Oriented projective geometry: a framework for geometric computations.* Academic Press, Inc., 1st. edition, 1991.