08/01/12 Pd ユーザ会＠新大阪用資料: David Shimamoto (gyokimae at h o t m a i l d o t c o m) GEM 編

前書き

このドキュメントの内容は、すべて著者の執筆時点での理解をまとめたものであり、正確さを保証するものではない。

概要

GEM とは、Pd にグラフィック機能を追加するライブラリでは、おそらく最もメジャーなもの。主な機能は以下の通り。

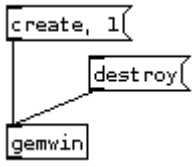
- 3D-CG の描画
- obj 形式の 3D モデルの読み込み
- 3D モデルへのテクスチャ貼り付け
  →テクスチャに使用する素材は、画像ファイル、動画ともに可。
- マウス・キーボードの入力検知
- ビデオ入力の処理、および簡単な動き検知
- カラー変換・キーイング（ブルースクリーンで背景を抜いたりするアレ）

背景を少々

GEM の 3D は OpenGL ライブラリに依存している。どうやって思いついたのか不思議に思う仕様やパラメータの多くは、調べてみると実は OpenGL で提供されている機能だったから…というのものがほとんど。
また、OpenGL をサポートするビデオカードがあると何かと便利。

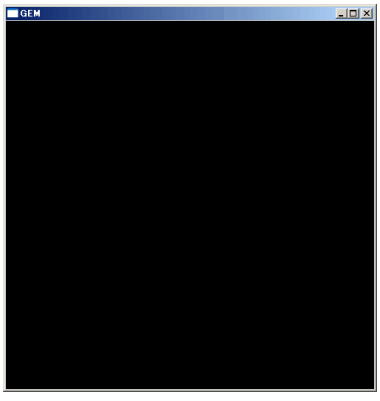
接続の基本(1) - GEM ウィンドウの作成

GEM を使用した描画はすべて別ウィンドウで行われるため、まずは作業用のウィンドウを作成するところから始めます。手順は簡単で、下図のように作成した[gemwin]オブジェクトに[create< メッセージを、次いでウィンドウを有効にするために [1< を送ります。あるいは例のように、カンマ区切りで一つにまとめても正しく解釈されます。

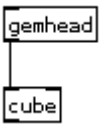


[create<だけを実行した段階ではウィンドウのみ表示され、描画はおろか、ウィンドウをアクティブにしても（選択しても）最上段にもってくることすらできません。GEM ウィンドウを破棄するには、例のように[destroy<メッセージを送信するか、単純に[gemwin]オブジェクトを削除します。

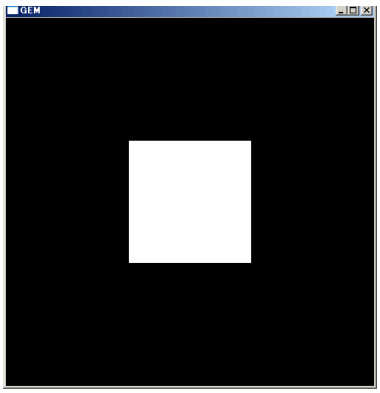
ここまでで、下の図のような作業ウィンドウができました。一旦 GEM ウィンドウを作成した後は、[gemwin]オブジェクトはパッチ内の適当なところに移動させて構いません。

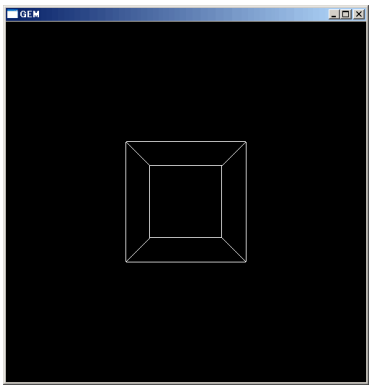


接続の基本(2) - オブジェクトの配置



次に、立方体を描画してみます。
まず、[gemhead]オブジェクトを作成ます。この[gemhead]というオブジェクトは、フレーム毎に「描画命令」を出力します。よって、これに立方体を描画する[cube]オブジェクトを接続すると、フレーム毎に立方体がウィンドウに描かれることになります（下図左）。
立方体がベタ塗りだと解りにくいので、立方体のフレームだけ描画したものが上図右にあります。奥行きがあるのがわかりますね。
（フレームのみ描画する方法については、練習がてら[cube]のヘルプをご参照下さい）





ここまでが、GEM におけるオブジェクトの基本的な描画方法です。この図形を変化させるには、[gemhead]から出力される「描画命令」と、最終的に命令を受け取るオブジェクトの間に、場所を移動する、回転するといった命令を追加するオブジェクトを挿入するわけです。

以下に変換に使用できる主なオブジェクトを、簡単にご紹介致します。

図形を回転させる

図形を平行移動する

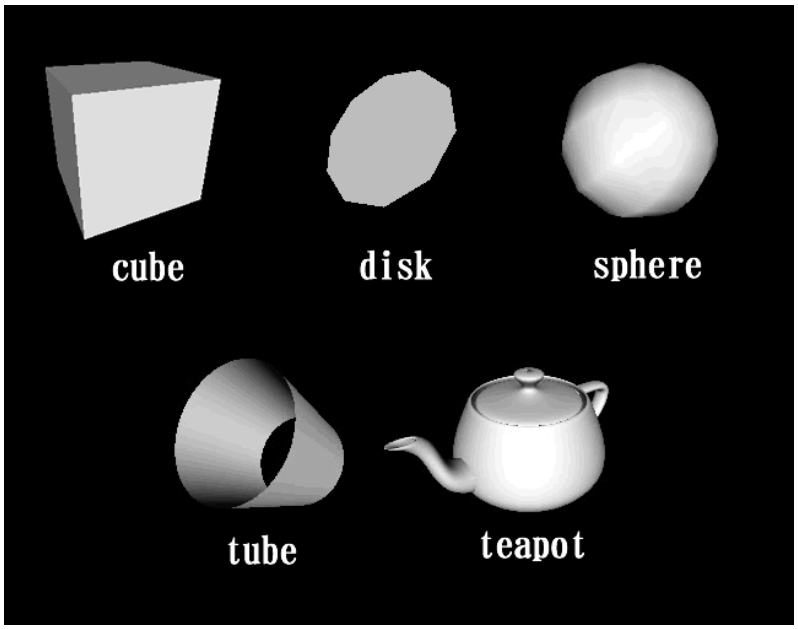
図形の色を変更する

[gemhead]の応用（中級）

前述のように、[gemhead]オブジェクトはフレーム毎に命令を出力します。例えばこの出力を[bang<
メッセージに接続すると、フレーム・レートで出力される bang を得ることができます。また、[gemhead]の出力を[repeat]オブジェクトに接続すると、繰り返した数だけのオブジェクトを１フレーム内に描画することも可能です。（描画毎に位置をずらす等の対処しなければ、重なって表示されるので見えませんが…）

プリミティブな形状

GEM には簡単に立体図形を描画できるオブジェクトがいくつかあります。以下にご紹介する以外の図形を扱うには、頂点の空間座標を順次入力して描画するか、外部の３Ｄモデリング・ツール(Blender 等)で作成した obj 形式のファイルを読み込む方法があります。



さて、これまでに描画した図形を「平行移動させてみよう」「回転させてみよう」といったチュートリアルはインターネット上に複数ありますので、今回は一足飛びにパーティクルの説明に進みます。基本的な図形を単独で扱うのはそんなに難しくありません。この時点では以下の点だけご理解いただければ結構です。

- 図形の描画命令は[gemhead]オブジェクトがフレーム毎に出力するものを指定する。
- [gemhead]の出力を、最終的に図形を描画する機能を持ったオブジェクトに接続することで、画面上に図形が描画される。
- 上記二つのオブジェクトの間に、描画する位置、回転などを変更するオブジェクトを挿入することができる。

備考：
上記の図中の図形はすべて、[rotate]オブジェクトで回転を加えてあります。
デフォルトでは上手の図形は、すべて単色でベタ塗りされたかたちで表示されます（白い図形が、最大の明るさで照らされている状態）。図のように陰影を加えるには、GEM ウィンドウ自体の「光源処理」機能を有効にして、環境光を設置します。光源処理を有効にするには、その旨を伝えるメッセージを[gemwin]に送信します。また、環境光を設置するには[world_light]オブジェクトを作成します。各オブジェクトの使用方法については、[gemwin] [world_light]のヘルプをそれぞれ参照して下さい。（これらはすべて、OpenGL ライブラリの標準機能です）
ちなみに光源を一つも設置しないで光源処理を有効にすると、画面が真っ暗になりますよ！

パーティクルについて

パーティクルとは本来「微粒子」を意味します。転じてＣＧ界においては、ある程度の規則性とランダム性を持つ無数の点をコンピュータで作成し、各々に何かしら処理をほどこすテクニックのことを指します。

パーティクルの使用例：

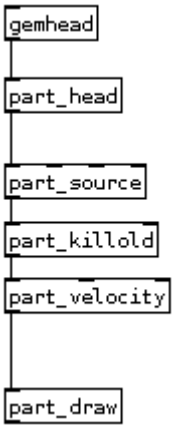
- 燃え上がる炎や、そこから舞い上がる煙の表現
- 舞い散る木の葉の表現
- 3D 格闘ゲームに見られる、技の弾道が光る処理

GEM の描画に関するライブラリが OpenGL に依存するのと同様、GEM はパーティクルの作成と管理に、Partical System API（入手先はドキュメント末尾のリンク集に記載）というフリーのライブラリを取り込んで使用しています。

余談：
Particle System API 自体は後方互換性を維持しながら地道にバージョンアップを重ね、現在では GEM が取り込んだバージョンにない機能が多数追加されています。しかし GEM は API のソースを直接改変して取り込むかたちをとっているため、これらの新機能を取り入れるにはまた改変が必要なものと思われます。

接続の基本

パーティクルを使用する際には、関連するオブジェクトの常套的な接続順があります。
先にお断りしておきますが、パーティクル関連のオブジェクトの多くが、初期パラメータを与えない限りほとんど機能しません。以下の例は大まかな流れを把握していただくためにパラメータを省略しているをあらかじめご了承下さい。



それでは、上から順に説明します。
他の図形を描画するときの例に漏れず、連続するオブジェクトの頂点には描画命令を出力する[gemhead]オブジェクトが必要です。続いて、パーティクル処理の開始を宣言する[part_head]オブジェクトを結びます。

続いて作成するのは、[part_source]オブジェクト。このオブジェクトがパーティクルを作成します。また、フレーム毎に発生させるパーティクルの数も、このオブジェクトに指示してやります。
[part_killold]は、一定時間後に古いパーティクルを消滅させます。例えばすべてのパーティクルが一方向に動作しているとしましょう。一つもパーティクルが消滅することなく増え続ければ、アプリケーションが管理するパーティクルは無限に増え続けるでしょう。そこで[part_source]で発生するパーティクルの数を、そして各パーティクルの寿命を[part_killold]で加減することで、使用するパーティクルの数をコントロールします。

先ほど、すべてのパーティクルが一つの方向に移動する場合を過程しました。パーティクルの移動方向を指定する方法はいくつかありますが、最も簡単なのは[part_velocity]オブジェクトで、初速度を与えてやる方法です。なにかしらの手段でパーティクルを動かさないことには、パーティクルは発生した地点で寿命が尽きるのを待つことになります。

今回はチェーンの最後に、[part_draw]オブジェクトを設置しました。これはパーティクルを点として描画するためのオブジェクトです。各パーティクルは空間内に一点の座標を持ちますが、描画す

る際には、何も点として表示する必要はありません。たとえば（別途オブジェクト挿む必要はあるものの）最後に先ほどと同じ[cube]オブジェクトを設置すれば、空間内のパーティクルの数だけ立方体が描画されます。

後日加筆：
会合の参加者はご覧になりましたように、パーティクルの初速度にもデフォルト値(sphere?)があるらしく、ブランクでもよいので[part_velocity]を挿さないと(これはこれでカッコイイけど)予期せぬ動作をするようです。

備考：
空間内に作成できるパーティクルの最大数は、[part_head]オブジェクトが管理しています。（デフォルトは 1,000）仮にパーティクルの発生するペースが消滅するペースより速い場合、上限値のパーテゥイクルが存在した時点で、新たなパーティクルは作成されなくなります。
するとどうなるでしょう？　しばらく後に消滅した数のパーティクルだけが新規に作成されるため、消滅を待つ間が挿まれ、パーティクルの発生頻度が不規則になります。すると、パーティクルの新規発生が間欠泉のように On/Off を繰り返します。
パーティクルの発生が滑らかになるよう維持するには、空間内のパーティクルが上限数に達しないよう適度な寿命を設定して下さい。

Domain について

さて、デフォルトではパーティクルは、空間内のある一点でしか発生しません。たとえば空間内の一定範囲内に、ランダムにパーティクルを発生させたい場合はどうすればよいでしょうか？

GEM のパーティクル・システム（具体的にいうと Partical System API）には、Domain という概念があります。これは、何種類からか選択できる基本図形と、その寸法を指定することにより、空間内の一定範囲を用意に指定する仕組みです。

以下のオブジェクトは、Domain で範囲を指定してできるオブジェクトの一部と、その作用について記します。

[part_source]　新規発生するパーティクルは、Domain 内のランダムな点を開始座標とします。

[part_velocity]　新規発生するパーティクルは、Domain 内のランダムな点に向かいます。この場合発生した地点との距離が遠いほど、初速度も速くなります。

[part_sink]　sink は、台所の「流し」を意味しており、パーティクルが存在できる範囲を指定します。この Domain が指す範囲を逸脱したパーティクルは直ちに消滅します。

各種 Domain

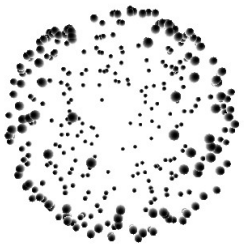
以下に、各 Domain の形状とパラメータを記載します。なお Domain の範囲は通常目に見えません。そこで、作成した Domain 内に、ランダムにパーティクルを発生させました。また、奥行きがわかるように個々のパーティクルは小さな球体として描画しています。小さい球体ほど、画面奥の方に位置しているのがおわかりになるかと思います。

Domain は寸法を指定する際、種類に応じて３～９つのパラメータを受け取ります。

'point'
空間内の一点のみを表す
arg.1　X 座標
arg.2　Y 座標
arg.3　Z 座標

'line'
空間内の直線を表す
arg.1-3　点 0 の X, Y, Z 座標
arg.4-6　点 1 の X, Y, Z 座標

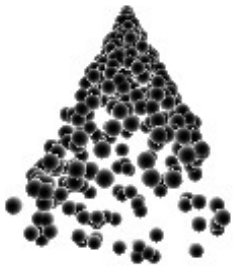
'rectangle'
平行四辺形を表す。
３つの頂点を指定すれば、最後の頂点は自動的に補完される。
arg.1-3　点 0 の X, Y, Z 座標
arg.4-6　点 1 の X, Y, Z 座標
arg.7-9　点 2 の X, Y, Z 座標



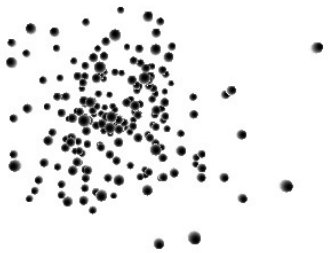
'sphere'
球体を表す。
arg.1-3　中心の X, Y, Z 座標
arg.4　外周の半径
arg.5　内周の半径

'box'
立方体を表す（画像では判りにくいですが…）
対角線上の頂点を二つ指定すれば、残りの頂点は自動的に補完される。
arg.1-3 点 0 の X, Y, Z 座標
arg.4-6 点 1 の X, Y, Z 座標

'cylinder'
円筒形を表す（これも画像では判りにくいですが…）
8 つ目のパラメータは任意。０より大きい場合、芯が空洞になる。
arg.1-3　点 0 の X, Y, Z 座標
arg.4-6　点 1 の X, Y, Z 座標
arg.7　外周の半径
arg.8　内周の半径



'cone'
円錐を表す（これも画像では判りにくいですが…）
8 つ目のパラメータは任意。０より大きい場合、指定した径の空洞をもつ。
arg.1-3　点 0 の X, Y, Z 座標
arg.4-6　点 1 の X, Y, Z 座標
arg.7　　底辺の半径
arg.8　　空洞の半径



'blob'
ランダムな範囲？詳細は不明…
arg.1-3　中心の X, Y, Z 座標
arg.4　　Standard deviation (??)

'plane'
ユークリッド平面を表す。
点０を通り、点０と点１を結ぶ直線に垂直な平面。
arg.1-3　点 0 の X, Y, Z 座標
arg.4-6　点 1 の X, Y, Z 座標

注意：[part_source]オブジェクトに対しては点０の座標しか返さないため、point と同じ挙動になる。

'disc'
円盤を表す。
上記 plane 上に円盤を描く。
arg.1-3　点 0 の X, Y, Z 座標
arg.4-6　点 1 の X, Y, Z 座標
arg.7　　外周の半径
arg.8　　内周の半径

参考リンク集

OpenGL (http://www.opengl.org/)

Particle Systems API (http://www.particlesystems.org/)

GEM for MaxMSP (http://gem4mac.sourceforge.net/)

Blender (http://www.blender.org/)
フリーの 3D モデリングツールです。

（不確定な後日追記）

パーティクルを Geo としてレンダーする際の注意点

個々のパーティクルを[cube][sphere]といった geo でレンダリングする際、二通りの方法があるようです。

[<パーティクルに関するチェーン>]
 |
[part_render]
 |
[cube]

と接続すると、大抵期待通りに動作しますが、コンソールの方にはエラーメッセージが連続して出力されます（文字化けするため内容は読めない）。

以下に示す異なる方法ではこのようなエラーは発生されません。

[<パーティクルに関するチェーン>]
 |
[part_info]
 |
[separator]
 |
[<各種変換>]
 |
[cube]

part_info は、個々のパーティクルを個別のチェーンとして扱うようで、[separator]を忘れると以降に挿入する各種変換が複数のパーティクルで共有されてしまい、期待通りの動作をしません。

また、[part_info]は色情報、位置情報、etc を一旦 GemList から分離します。これらを復元するには[<各種変換>]とした個所に[translate] [color]等を挿入し、[part_info]の Outlet3, 4 から取り出した数値で再度定義してやる必要があります。

逆に part_info を自作の table(array)と組み合わせることにより、各パーティクルに個別の色や回転角等を定義できるので、通常より part_info を挿入する習慣を身につけておけば後々楽かも知れません。

この辺りに関する正統なメソッド、また内部で OpenGL ライブラリがどのように扱われているにつきまして、識者からの情報をお待ちしております。