IMAL – Stage 3D temps réel et modèles physiques

Animateur: Nicolas Montgermont nm@chdh.net

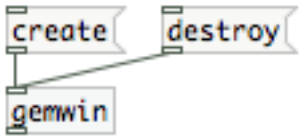
Mardi 3 août 2010 AM

Il faut d'abord installer ***PureData Extended*** en version 0.4 ou supérieure, à décharger sur : http://puredata.info/downloads.

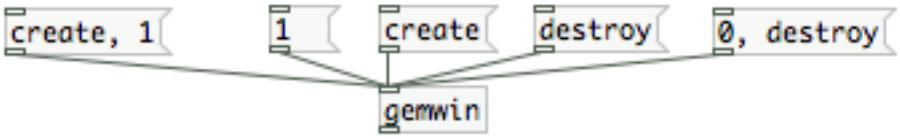
Pour tester si **GEM** (Graphics Environement for Multimedia) est bien installé, dans PureData, mettre un objet ***gemhead*** et voir s'il est accepté (pas de cadre rouge autour de l'objet).

Le premier objet qu'on va ensuite étudier est: ***gemwin***

gemwin peut recevoir les deux messages ***destroy*** et ***create*** qui permettent de créer et détruire la fenêtre de rendu. Dans le fichier ***Exercice1.pd*** on trouve le premier exemple ci dessous:



Les messages ***1*** et ***0*** appliqués à ***gemwin*** servent respectivement à activer et désactiver le rendu. On peut combiner les messages en enchaînant les messages séparés par une virgule: ***create, 1*** par exemple.



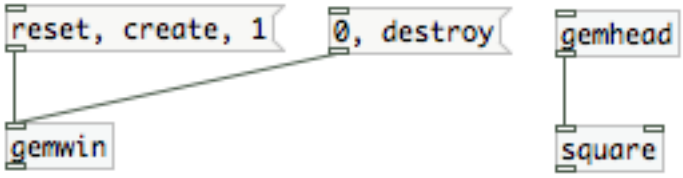
Remarque: dans PureData, Pomme-E permet de passer du mode édition au mode exécution

On va maintenant dessiner des éléments (figures) dans la fenêtre (fichier ***Exercice2.pd***).

Pour cela on utilise ***gemhead*** associé à un objet qui permet de réaliser un figure, par exemple l'objet: ***square***

Attention: si rien n'apparait dans la fenêtre, il faut peut-être réinitialiser l'objet ***gemwin*** avec le message ***reset***. Donc, pour créer une fenêtre, il est habituel d'envoyer le message combiné: ***reset, create, 1***

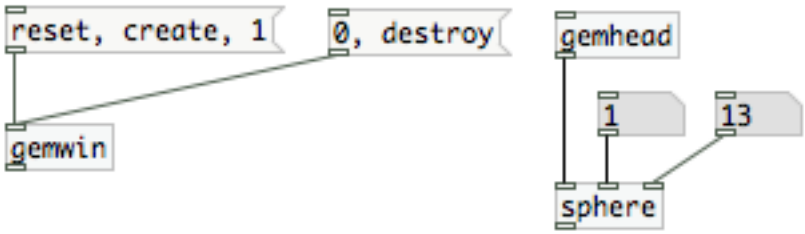
Attention: ***reset*** est obligatoirement envoyé avant le ***create***.



Sur la deuxième entrée de ***square*** on peut indiquer la taille du carré. On met donc un ***number*** dont on modifie la valeur avec la souris combiné à la touche *shift* si on veut plus de précision (décimales).

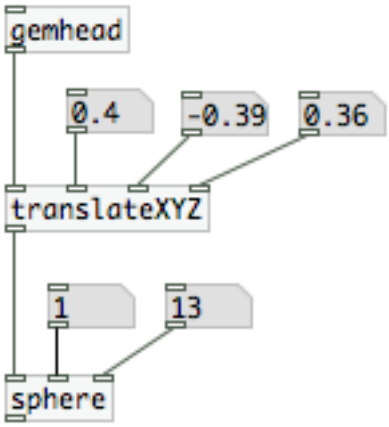
Voyons maintenant les objets ***cube***, ***circle*** et ***sphere***.

Les formes dans *OpenGL* sont dessinées avec des segments (vecteurs). Ceci est très visible dans l'objet ***sphere*** qui permet en fonction du nombre de segments choisis (la troisième entrée) de faire différentes formes (voir ***Exercice2.pd***).



Remarque: Dans l'aide, on a ***Browser*** qui permet d'avoir de l'aide, des informations mais surtout des exemples, des patchs de base. Par exemple, dans *examples/Gem/01.Basic* ...

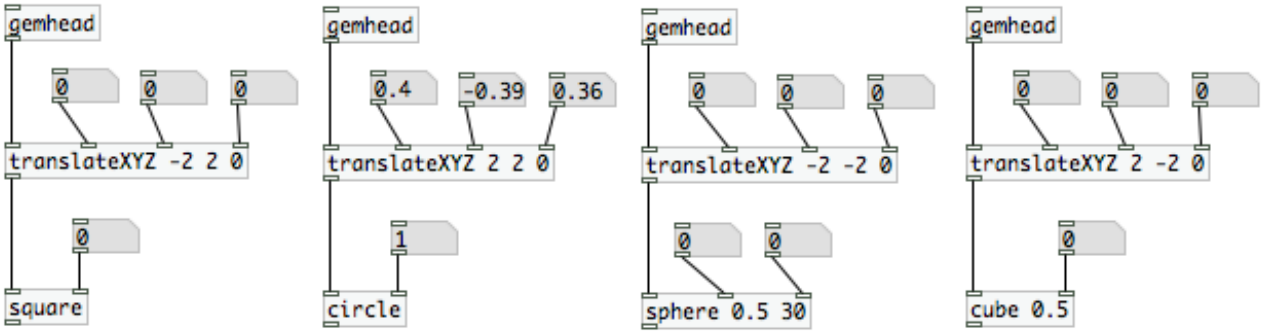
On passe à l'***Exercice3.pd*** dans lequel on va modifier **la position** des figures. On utilise pour cela ***translateXYZ***. Il s'insère entre ***gemhead*** et l'objet correspondant à la figure.



Attention: **le système de coordonnées** de base (par défaut) est centré au milieu de la fenêtre (caméra en position de base) avec (4, 4) comme coordonnées sur le bord en haut à droite.

-4, 4	4, 4
0, 0	
-4, -4	4, -4

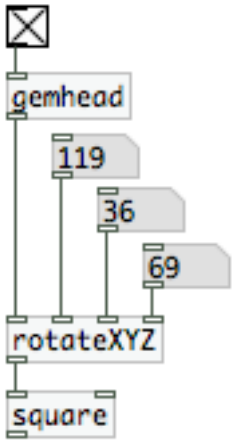
Pour afficher plusieurs figures, il faut, pour l'instant (on verra plus tard qu'on peut procéder autrement), mettre plusieurs ***gemhead***.



Pour réaliser **un sous-patch**, on le crée d'abord avec un objet ***pd*** suivi du nom qu'on veut lui donner (***deplacement***, par exemple). Ceci est réalisé dans ***l'Exercice4***.

Ensuite, on va désactiver les ***gemhead*** (pour gagner en puissance de calcul ou pour ne plus afficher une portion du dessin, par exemple) en lui envoyant un zéro (via un ***toggle***, par exemple)

Pour réaliser **une rotation** de l'objet on utilise l'objet ***rotateXYZ***.
Attention: ici on ne bouge pas la caméra, c'est bien l'objet qui tourne.



Ensuite, on va insérer une translation dans la chaîne.

Attention: une translation suivie d'une rotation n'est pas équivalente à une rotation suivie d'une translation. On peut visualiser la différence dans ***l'Exercice4.pd***.

On va maintenant dans l'***Exercice5.pd*** tester l'objet ***scaleXYZ*** qui permet un redimensionnement ou un changement de proportions.

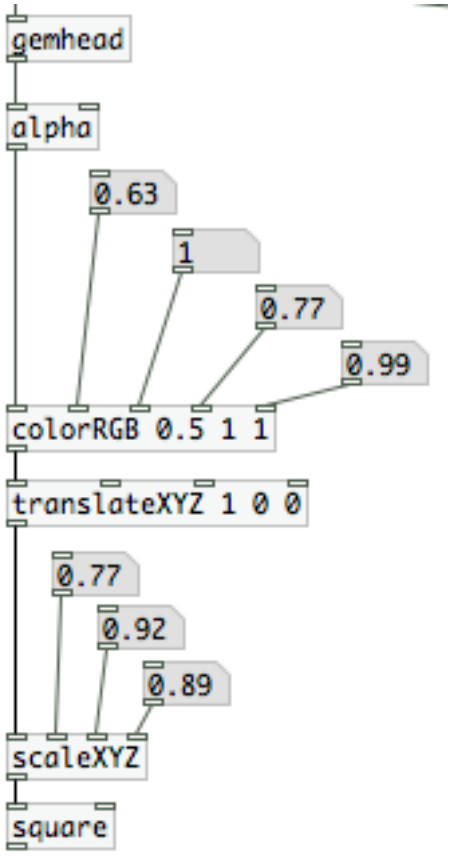
Même remarque que pour la rotation, sa position dans la chaîne influence son comportement. Les facteurs de ***scale*** sont multiplicatifs, donc zéro réduit la forme à rien.

Pour terminer cette première partie, on va mettre les figures en **couleurs** avec l'objet ***colorRGB***.

Cet objet permet de choisir la couleur de l'objet avec les trois composantes rouge, verte et bleue comprises entre 0 et 1.

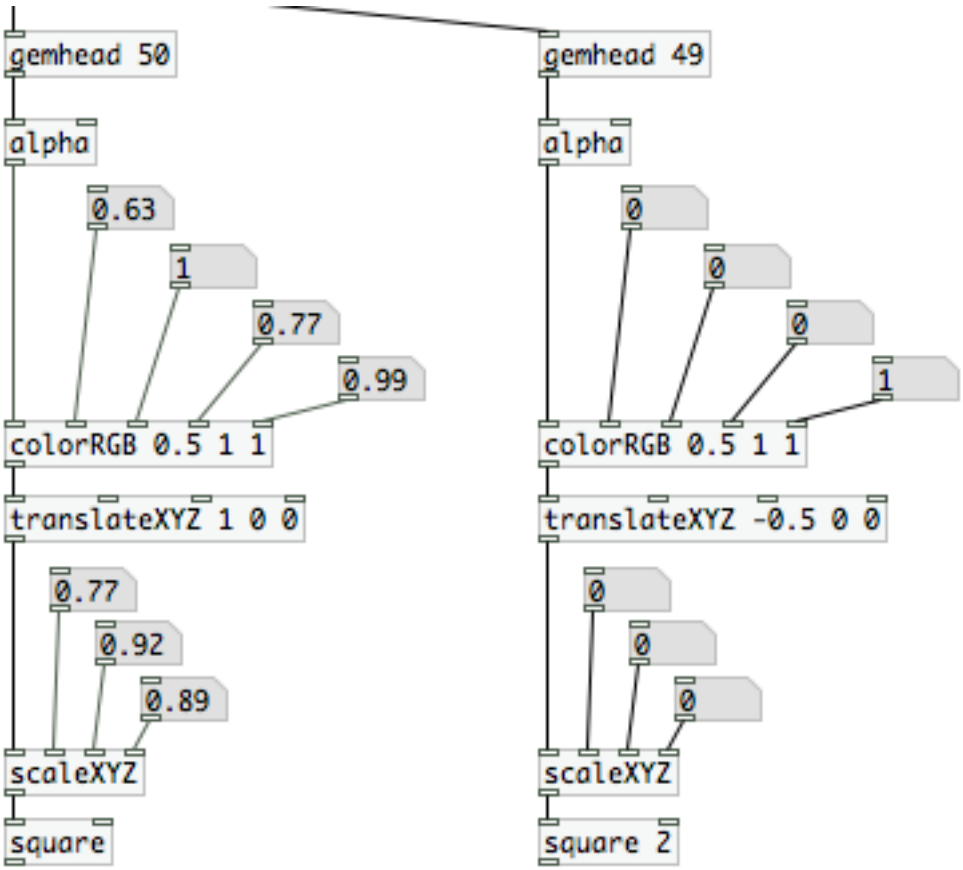
On peut voir un exemple de son utilisation dans l'***Exercice6.pd***.

La transparence est gérée par le dernier paramètre de l'objet ***colorRGB***. Ce paramètre est également compris entre 0 (transparent) et 1 (opaque). Il n'est à priori pas calculé (il consomme des ressources), il faut donc insérer un objet complémentaire: ***alpha*** (à insérer n'importe où dans la chaîne du ***gemhead***).

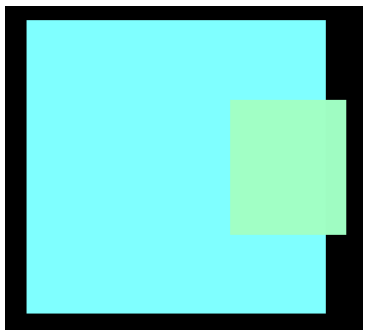


Attention: En fonction de l'ordre de rendu, une figure peut être au dessus ou en dessous d'une autre.

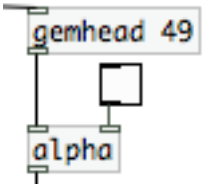
Il faut donc spécifier cet ordre « à la dure » grâce au premier paramètre du ***gemhead*** (par défaut 50).



Dans ce dernier exemple, le carré créé à gauche est dessiné après celui de droite ce qui fait que le deuxième apparait par dessus le premier.



On peut changer le mode de transparence sur l'objet ***alpha*** (la manière dont la transparence est calculée) avec le second paramètre (inlet de droite) qui prend 0 ou 1.

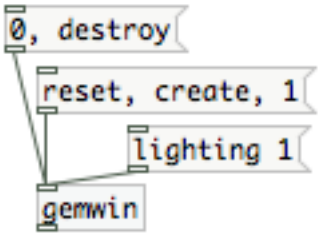


Pour plus de détails, se reporter à l'aide.

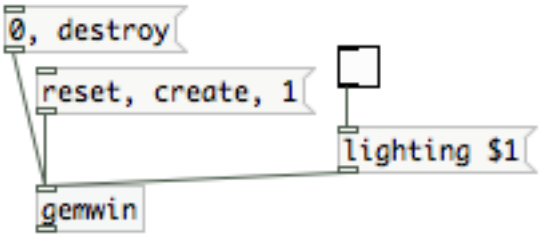
Réalisation de l'exercice imposé numéro 1 dans le fichier ***impose1.pd***

Mardi 3 août 2010 PM

Pour bien visualiser la 3D, il faut éclairer la scène. **Les lumières** sont gérées par différents objets. Voyons d'abord les messages ***lighting*** qui à la base sont à zéro (consomme des ressources). On envoi comme d'habitude un 1 pour l'activer (mais précéder par le message lui-même, soit au final : « ***lighting 1*** »).



On peut l'activer ou le désactiver plus simplement grâce au message ***lighting $1*** précédé d'un objet ***toggle***.



Ensuite on utilise l'objet ***world_light*** qui simule une lumière à l'infini (tous les rayons parallèles).

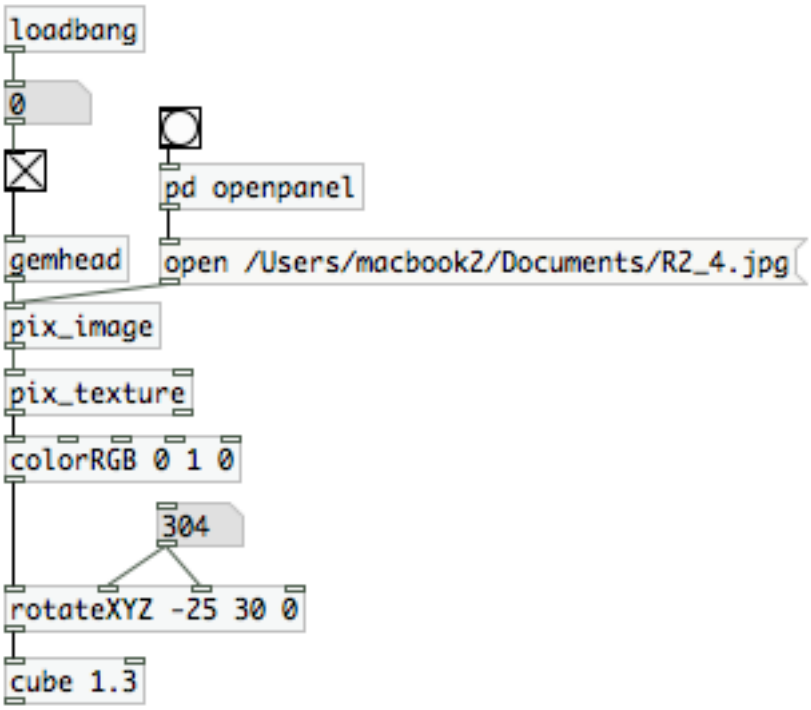
L'objet ***light*** simule une lumière ponctuelle qui émet dans tous les sens (type ampoule).

Il existe un dernier objet plus complexe et aussi plus complet ***spot_light*** qui permet de contrôler le cône de lumière.

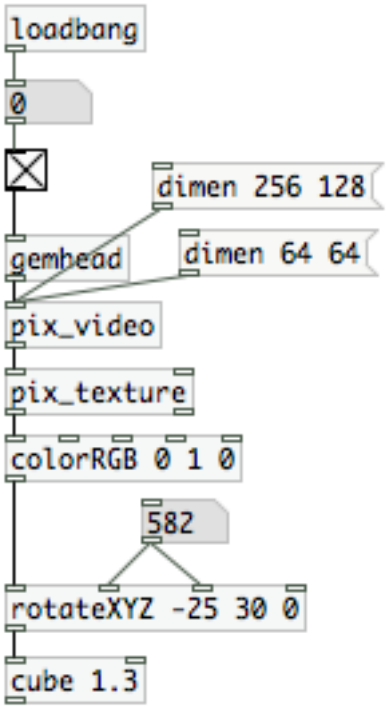
On trouve un exemple d'utilisation de ces objets dans l'***Exercice7.pd***.

Remarque: le mode ***debug*** des ces objets permet de visualiser la source lumineuse pour mieux prévisualiser l'effet et éventuellement plus facilement la positionner.

Les textures sont plaquées à partir d'un objet: ***pix_texture***. Mais avant de plaquer la texture il faut charger un BMP à partir de l'objet ***pix_image***, par exemple. On peut même après le premier objet modifier la couleur avec un ***colorRGB*** que nous avons déjà vu. L'***Exemple8.pd*** illustre leur utilisation.



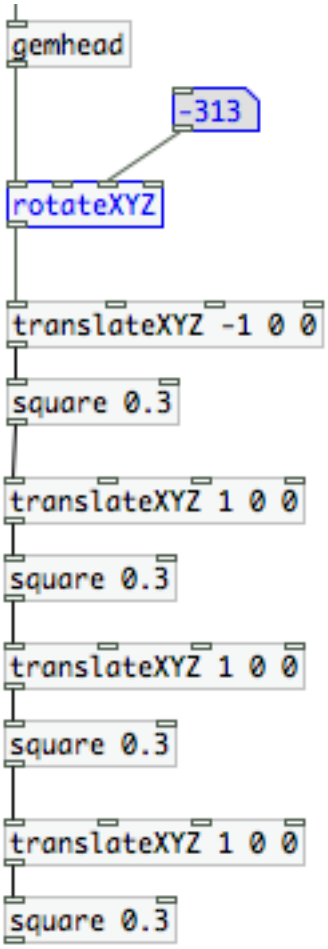
Les textures peuvent provenir d'une webcam. Pour ce faire c'est l'objet ***pix_video*** qui est utilisé. On peut tester un exemple d'utilisation de ces textures dans l'***Exercice9.pd***.



Dans l'***Exercice9bis.pd***, il y a une variante avec la texture plaquée sur une sphère qui correspond aussi à l'exercice imposé numéro 2: ***Impose2.pd***.

Dans l'***Exercice10.pd***, on retrouve des variantes de texture qui utilisent des séquences vidéos à la place de la webcam. Les deux objets principaux sont ***pix_film*** et ***pix_movie***.

On a toujours pour l'instant mis les ***gemhead*** en parallèle pour dessiner plusieurs objets. On pourrait en fait enchaîner des opérations. L'intérêt est d'appliquer des transformations globales (***rotateXYZ*** dans l'exemple suivant) sur des groupes d'objets (les différents ***square***).



On peut, au contraire ne pas accumuler les transformations en utilisant un objet **separator** qui permet d'ordonner un traitement en parallèle.

On pourrait donc produire le même résultat que précédemment mais en changeant les valeurs du ***translateXYZ*** pour chaque ***square*** en parallèle (attention de ne pas oublier l'objet ***separator*** pour chaque « branche »). Voir ***Exemple11.pd***.

L'intérêt est évident pour la transformation d'un ensemble de formes en même temps mais également pour le chargement des textures qui ne doivent, dans ce cas, qu'être réalisé qu'une seule fois.

La journée se termine avec le dernier exercice imposé, le numéro 3. Ma solution est dans le fichier ***Impose3.pd***. Il y a également deux versions un peu différentes proposées par Nicolas.

Mercredi 4 août 2010 AM

Il existe d'**autres formes** (figures) que celle que nous avons vu hier.

Pour commencer simplement on peut découvrir que ces formes ont un paramètre commun: ***draw***. Il permet d'afficher les figures en pointillés, en fil de fer, etc. Il suffit d'envoyer dans les objets de dessin (***cube***, ***sphere***, ***circle***, ...) un message de type: ***draw line*** ou ***draw fill*** ou ***draw point***.

Voir l'exemple de Nicolas: ***4.a.draw.pd***.

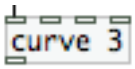
On peut aussi découvrir de nouvelles formes avec les objets ***rectangle***, ***triangle***, ***disk***, ***torus***, ***cone***, ***cuboid***, ***cylinder***, ***tube***.

Attention: les dimensions des formes sont toujours mesurées entre le point (0, 0) et le côté. Par exemple, pour une sphère, la dimension donnée est le rayon, ce qui parait logique mais pour un carré, la dimension donnée est le demi côté. Même remarque pour le rectangle, triangle, ...

Remarque: le paramètre ***draw*** vu précédemment peut également être utilisé pour toutes ces nouvelles formes.

On retrouve un récapitulatif et un exemple de chacune des figures dans mon ***Exercice12.pd*** et dans le ***4.b.formes_definies.pd*** de Nicolas.

D'autres formes nécessitent un nombre de paramètres variables. L'objet ***curve***, par exemple, prend en argument le nombre de points qui constitue la courbe. Une fois placé, le nombre de point défini fait apparaître autant d'*inlets* que nécessaire. Ci-dessous on voit un ***curve*** définit avec 3 points et donc 3 inlets.



Les objets ***curve3d*** et ***polygon*** fonctionnent de la même façon. Il existe une référence *OpenGL* sur le site http://www.glprogramming.com/red/, les formes sont dans le chapitre 2 ici http://www.glprogramming.com/red/chapter02.html#name2 ce qui permet d'appréhender tous les paramètres complexes de ces formes complexes.

Remarques: un autre lien OpenGL:
http://www-evasion.imag.fr/Membres/Antoine.Bouthors/teaching/opengl/

On peut trouver dans mon ***Exemple13.pd*** une application du ***curve*** avec trois points dont un mobile.

L'***Exercice14.pd*** réalise une petite application qui dessine une sinusoïde en mouvement à partir d'un ***curve*** à 5 points dont deux mobiles.

L'objet ***curve3d*** est complexe mais intéressant pour créer des grilles, des surfaces complexes, ... à tester dans l'aide.

L'objet ***polygon*** est très semblable à ***curve*** mis à part la fait que le tracé est rectiligne. On retrouve dans l'***Exercice15.pd*** la réalisation d'une étoile avec cet objet ***polygon***.

Remarque: le paramètre ***draw line*** permet de « refermer » le polygone sur le premier point. Par contre, le paramètre ***draw linestrip*** arrête le tracé sur le dernier point.

Un exemple d'utilisation de ces trois derniers objets (***polygon***, ***curve*** et ***curve3d***) sont repris dans le fichier ***4.c.formes_a_construire*** de Nicolas.

Les objets qui permettent d'**afficher du texte** sont ***text2d***, ***text3d***, ***textextruded*** et ***textoutline***.

Le premier de ces objets (***text2d***) permet d'afficher un texte, en modifier sa taille (sur l'*inlet* 2), son contenu (avec le paramètre ***text***), changer le set de caractère, activer/désactiver l'antialiasing sur le texte et pas sur la totalité de la fenêtre (avec le paramètre ***alias***) et justifier (paramètre ***justify***).

Paradoxalement, l'objet ***text3d*** est moins gourmand en ressource que son prédécesseur. Cependant il ne permet pas d'antialiasing « local ». Il faut donc appliquer cet antialiasing au niveau de la fenêtre (réalisé sur la carte graphique) à sa création (paramètre ***FSAA 4*** à appliquer sur ***gemwin***).

Les deux derniers objets ***textextruded*** et ***textoutline*** permettent respectivement d'afficher du texte « avec une épaisseur » et « non rempli ».

Mon ***Exemple15.pd*** et le fichier ***4.d.texte.pd*** de Nicolas illustrent ces possibilités.

Mercredi 4 août 2010 PM

Rappel: une **abstraction** désigne un patch (fichier avec l'extension .pd) qui réalise une fonction destinée à être utilisée au sein d'un autre patch.

Attention: Pour que cela fonctionne l'*abstraction* doit se trouver dans le même répertoire que le patch hôte.

Pour réaliser une *abstraction*, il suffit de créer un patch qui comporte des ***inlet*** et ***outlet*** (comme dans les *sous-patch*) et de le sauvegarder avec un nom « parlant » (correspondant à la fonction réalisée). Ensuite, dans un autre patch, pour « appeler » l'*abstraction*, il suffit de créer un objet qui porte le nom du fichier (sans l'extension .pd).

On peut définir des arguments dans une abstraction en les spécifiant dans l'abstraction avec $1, $2, etc. Ceci permet de personnaliser ses abstractions.

Les abstractions, les objets de type « texte » vont trouver une application dans un exercice imposé récapitulatif ***Impose4.pd*** qui réalise un espèce d'effet « matrix » (la pluie de lettres vertes qui tombent vers le bas de l'écran).

Remarques:

Lien vers archives mailing-list:
http://www.mail-archive.com/pd-list@iem.at/maillist.html
Lien vers forum:
http://puredata.hurleur.com/
Lien vers forum francophone:
http://codelab.fr/pure-data

L'objet ***bytes2any*** sert à convertir un nombre en caractère (conversion *ASCII*). Mais comme il fait partie d'une librairie particulière, il faut importer cette librairie dans le patch avec l'objet ***import*** suivi du nom de la librairie: ***moocow***.

Jeudi 5 août 2010 AM

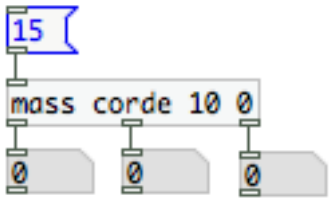
La modélisation physique dans Pure Data permet de simuler des comportements physiques élémentaires. Pour commencer on va simplement créer des masses et les lier entre elles.

Attention: les objets que l'on va utiliser spécifient leur nombre de degré de liberté. On aura, par exemple, les objets ***mass***, ***mass2D*** et ***mass3D*** pour les trois déclinaisons (un axe, deux axes, trois axes) du même objet ***mass***.

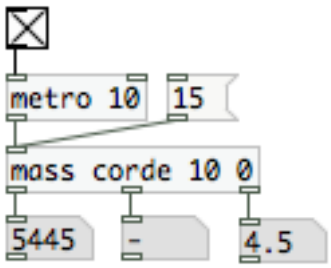
Pour vérifier si la librairie est correctement installée (à priori, il n'y a rien à faire, c'est en standard dans Pure Data Extended) il suffit d'aller dans le *browser* -> exemples/pmpd/01_basics.pd et voir si l'animation du dessus (les lettres PMPD qui tombe) fonctionne.

Pour la première masse on utilise l'objet ***mass*** qui comporte trois arguments, son nom, sa masse et sa position initiale.

On peut lui appliquer une force à l'aide d'un message, ce qui provoquera une modification de ses sorties: sa position, la force résultante appliquée sur la masse et sa vitesse.

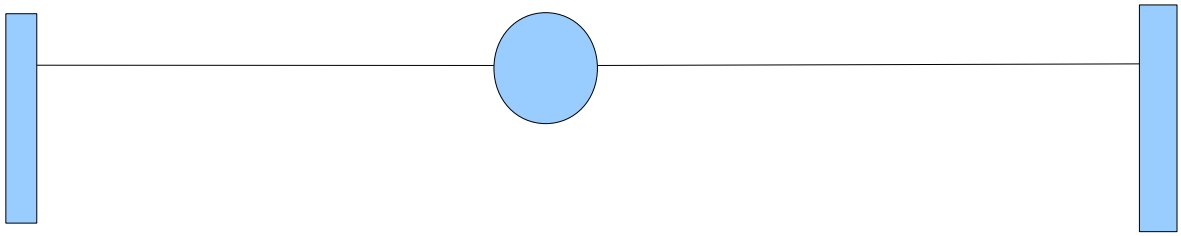


Rien ne se passe si on ne raccorde pas un ***metro*** qui envoi régulièrement des bang sur l'entrée de l'objet mass pour recalculer le mouvement.

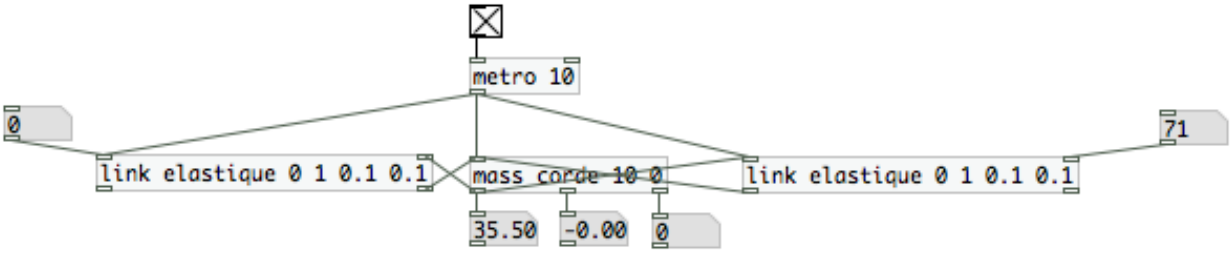


Ensuite, on va lier ces masses avec un objet ***link***. Cet objet possède deux *inlet* de position et deux *outlet* de force.

Finalement, pour réaliser ce modèle simple comportant une masse, deux élastiques et deux points d'attaches mobiles:

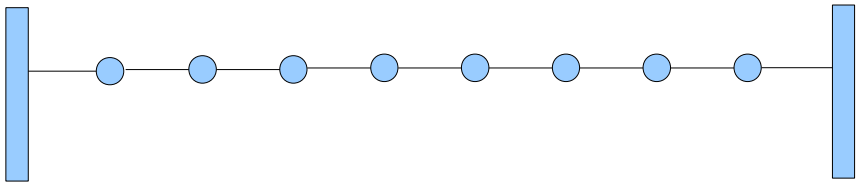


Il suffit de créer le petit patch suivant :



Ce premier exemple est illustré dans l'***Exercice17.pd*** qui montre grâce au slider le mouvement de la masse en fonction de la modification de la position des points d'ancrage sur les deux côtés.

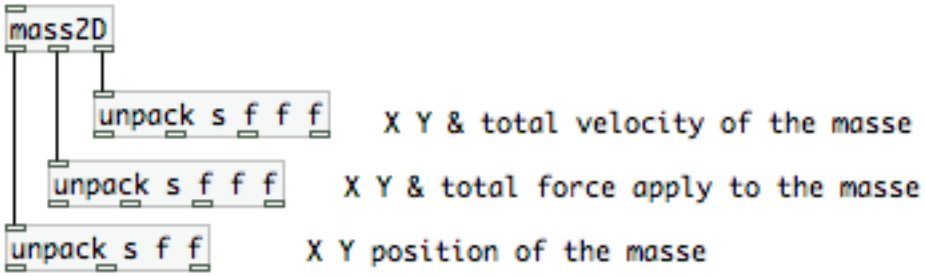
Nous allons ensuite simplement enchaîner plusieurs masses de la même façon de manière à créer une chaîne qui pourrait s'apparenter à une corde.



Cette solution se trouve dans la résolution de Nicolas ***5.corde.pd***. Nous allons transformer cette solution pour visualiser dans GEM le résultat. C'est assez simple, il suffit de transformer l'intervalle de valeur **0 -> 127** en un intervalle **-4 -> 4**. On réalise également une abstraction de l'affichage d'une masse de manière à simplifier la visualisation du patch. On peut finalement y ajouter la gestion de la souris dans la fenêtre GEM grâce à l'objet ***gemmouse***. La solution complète se retrouve dans l'***Exercice18.pd***.

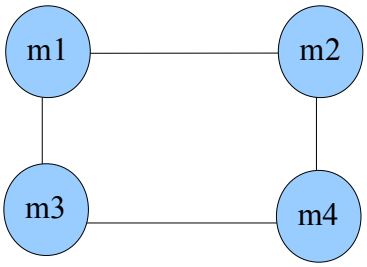
L'étape suivante consiste à réaliser un système à deux dimensions. Comme nous l'avons signalé plus tôt, les objets sont particuliers à cette deuxième dimension: ***mass2D*** et ***link2D***.

Le premier objet, une fois inséré dans un patch se présente de la même façon que l'objet ***mass***. La différence réside dans la manière de récupérer les valeurs en sortie qui représentent toujours dans l'ordre: la position, la force résultante appliquée sur la masse et la vitesse. Mais comme nous sommes en 2D chacune de ces informations sont données sous forme de ***listes***. Il faut donc utiliser un ***unpack*** pour visualiser les valeurs en X et Y.

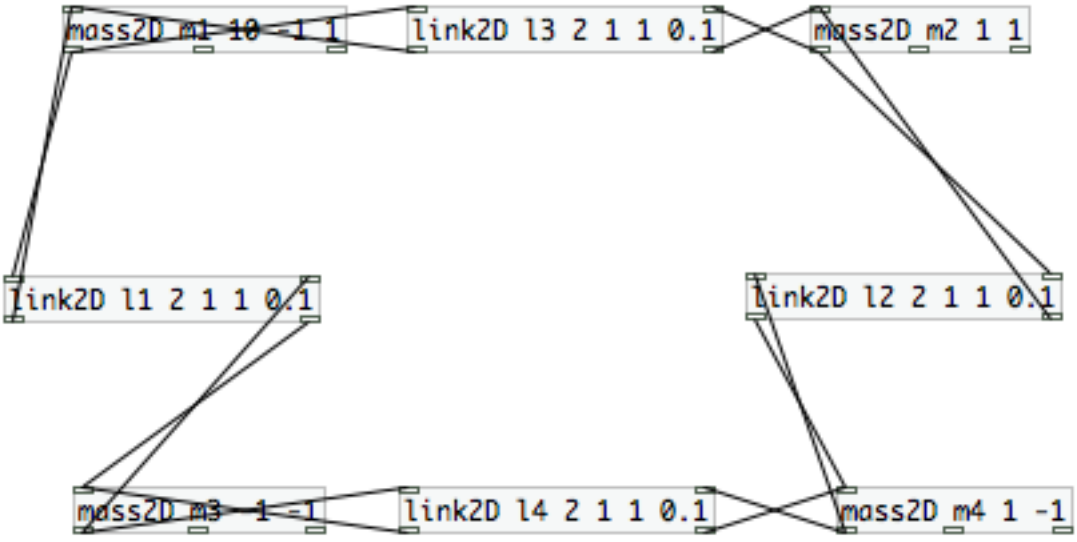


En entrée, c'est pareil, les messages doivent avoir la forme: ***force2D 10 20***

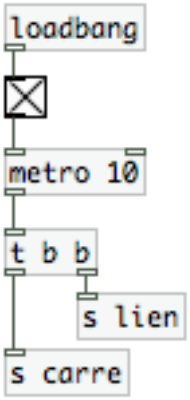
Le fonctionnement est identique au procédé vu plus haut. Pour réaliser ce petit ensemble:



Il suffit de réaliser le patch suivant:



Attention: ne pas oublier, comme toujours de mettre un ***metro*** qui enverra un ***bang*** régulièrement pour rafraichir le calcul.



Pour faciliter cela on peut nommer tous les ***link2D*** de la même manière (lien, par exemple) et toutes les ***mass2D*** avec le même nom également (carre, par exemple) pour n'avoir qu'à envoyer via l'objet ***s*** le message vers tous les objets directement et surtout avec moins de « fils à brancher ». Ceci est réalisé dans l'***Exemple19.pd***.

Pour représenter les masses, c'est la même technique que dans les exemples précédents. Par contre ici, nous allons représenter les liens entre les masses avec des lignes (objet ***curve 2***).

Attention: On ne peut pas récupérer les positions des extrémités des liens. On doit donc les obtenir directement en fonction des positions des masses auxquels ils sont attachés.

On va ensuite **définir des bornes aux masses**. Ce sont en fait les coordonnées des bords qui sont envoyées grâce aux messages ***setXmin*** et ***setYmin***. Cette dernière opération va provoquer des rebonds sur les limites.

Pour terminer, on va ajouter deux liens « plus rigides » en diagonale de manière à conserver un carré quelques soient les forces appliquées.

La solution finale est dans le fichier ***6.carre.pd*** de Nicolas.

On va maintenant mettre de **la gravité** dans le système en envoyant une force en continu (simplement en utilisant un ***metro***) vers toutes les masses.

On peut aussi utiliser un objet ***iAmbiant2D*** qui est plus complexe à mettre en oeuvre mais qui a aussi plus de possibilités. Il peut, par exemple, limiter l'action de ces forces à une zone de l'espace, ajouter de l'aléatoire, etc.

Jeudi 5 août 2010 PM

On va un peu aborder **les interfaces graphiques** dans Pure Data.

Remarques:

- voir ***qeve*** et ***le hangar*** à Barcelone (centre multimédia)
- voir ***pdp*** une autre librairie destinée à la vidéo
- voir et X11 un serveur vidéo disponible d'origine sur Mac

On peut déjà faire pas mal de choses en personnalisant (taille, couleur, ...) les objets de base tels que les ***radio***, ***toggle*** et ***sliders***. Le ***canvas*** est un objet purement graphique qui permet de créer une surface colorée qui encadre, « rassemble » les « fonctions » du patch.

Attention: Les ***canvas*** viennent se placer au dessus des objets déjà présents dans le patch, il vaut donc mieux les placer en premier lieu.

Dans les propriétés du sous-patch, on peut cocher l'option ***graph on parent*** qui permet d'afficher une partie du sous-patch dans le patch hôte.

L'objet ***key*** permet de récupérer les touches enfoncées sur le clavier.

La parenthèse interface graphique terminée, nous allons réaliser l'exercice imposé numéro 6 qui trouvera sa solution dans ***Impose6.pd***.

Pour **installer des objets complémentaires** dans Pure Data, la meilleure solution est d'aller les déposer dans ***Users/votrenom/Bibliotheque/Pd***. Ces fichiers en général ont l'extension ***pd_darwin***.

Dans notre cas, nous avons installé les trois objets ***msd***, ***msd2D*** et ***msd3D***. Les exemples sont dans le Browser -> ***examples/msd/msd/...***

On va pouvoir grâce à ces objets créer plus rapidement de grosses structures *masses-liens*.

Analysons le fonctionnement de l'exemple que l'on retrouve dans Browser -> ***examples/msd/msd/03_msdwave.pd*** pour le modifier et donner l'exercice imposé numéro 7 sauvegardé dans le fichier ***Impose7.pd***.

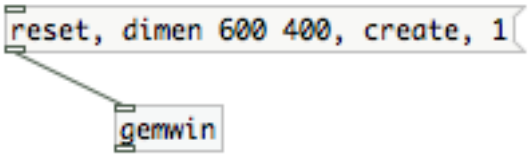
On trouve dans les *patch* d'exemple assez fréquemment le paramètre ***$0***. Il s'agit en fait du *handle* du patch, c'est-à-dire un identifiant unique du patch, sous-patch ou abstraction dans lequel on se trouve. C'est un moyen de s'assurer que les messages ne sont pas envoyés à d'autres *patch* qui seraient ouvert en même temps.

L'objet ***until*** est très intéressant pour créer des bang en rafale (***uzi*** dans Max). Si on désire créer une multitude de masses et de liens, on pourra donc créer simplement une structure du genre:

Vendredi 6 août 2010 AM

Nous reprenons l'exemple du Browser -> ***examples/msd/msd/03_msdwave.pd*** pour le encore le modifier et réaliser des variantes à l'exercice imposé numéro 7 sauvegardé dans les fichier ***Impose7_V2.pd***, ***Impose7_V3.pd***, ***Impose7_V4.pd***, etc.

Pour parfaire les fenêtres, il faut étudier **les différents paramètres de l'objet *gemwin***. Par exemple, on peut définir la dimension de la fenêtre avec le message ***dimen*** (avec la taille en pixels en X et en Y) qui doit être envoyé après le ***reset*** mais avant le ***create***. La fenêtre ne peut plus changer de taille après sa création.



Attention: le repère de base va changer si les proportions changent. En fait, verticalement le repère reste calé sur l'intervalle -4, 4 et c'est horizontalement que le repère sera étendu. Par exemple, si on crée une fenêtre d'une taille de 800 x 400, on aura le repère suivant:

-8, 4	8, 4
0, 0 -8, -4	8, -4

Le message ***fullscreen 1*** permet de donner tout l'espace disponible à la fenêtre et le message ***menubar 0*** fait disparaitre la barre de menu. Pour également faire disparaitre la bordure de la fenêtre on envoi le message ***border 0***.

Remarque: le ***fullscreen*** a parfois tendance à déformer les proportions. On lui préfèrera donc en général un redimensionnement suivi des ***menubar 0*** et ***border 0***.

Attention: une fois en mode « plein écran » , on ne peut plus d'atteindre les menus, il est donc prudent de prévoir une touche d'échappement qui permet de fermer la fenêtre avec la touche *ESC*, par exemple. Pour cela on utilise l'objet ***gemkeyboard*** qui récupère les touches enfoncées au clavier lorsque la fenêtre ***gem*** est active.

On peut également désirer déplacer la fenêtre ***gem***, sur un autre écran par exemple. Pour cela, on utilise le message ***offset -1024 -50*** envoyé avant la création de la fenêtre.

Pour éviter un passage de curseur de souris sur l'écran en pleine diffusion, il est prudent de le cacher avec le message ***cursor 0***.

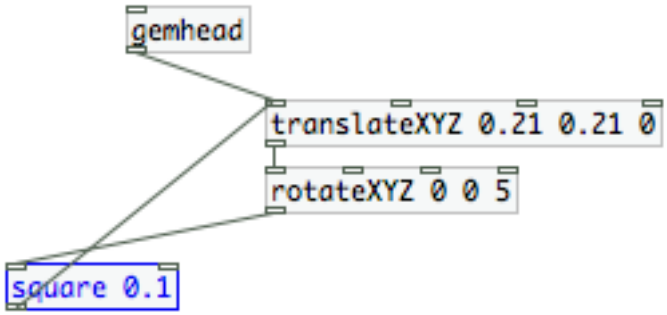
Le nombre d'images par seconde est géré par le message ***frame 25***, par exemple. La valeur par défaut est de 20. Ce paramètre est directement proportionnel à la demande de calcul. Il est donc très important de le régler au plus juste.

Un autre message intéressant est le message ***view*** qui permet de déplacer la caméra (le point de vue) mais il a déjà été vu précédemment. Nous avions également déjà évoqué le problème d'antialising qui est réglé au niveau de la carte graphique en envoyant le message ***FSAA 4***.

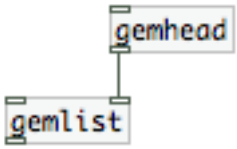
On peut utiliser la ***gemwin*** en « accumulation ». C'est à dire qu'on va écrire dans le *buffer* sans effacer les *frames* précédents. C'est peut consommateur de ressources et cela peut donner des effets intéressants. Pour appréhender ce message ***buffer*** on peut d'abord analyser l'exemple que l'on retrouve dans le *Browser* -> examples/Gem/04.pix/10.PixDataSimple.pd

Le fichier ***parametres_gemwin.pd*** de Nicolas récapitule ces fonctionnalités.

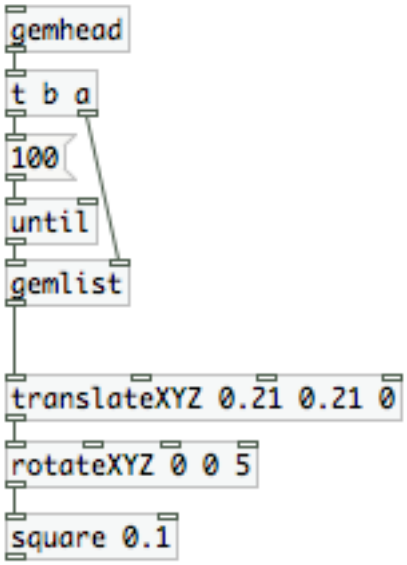
La récursion va faire boucler un ensemble d'opérations de bases sur elles-même.



Le problème avec cette récursion, c'est qu'elle est infinie et non contrôlée. On va donc plutôt procéder à une itération. Pour ce faire on va stocker le ***gemhead*** dans une ***gemlist***



et cette ***gemlist*** sera « *bangée* » par un objet ***until*** autant de fois qu'on le souhaitera (100 fois ici) pour créer l'itération.



On peut ensuite insérer dans la chaîne des variations de rotation, de translation, de couleur, d'échelle, etc. On retrouve un exemple dans mon ***Exercice22.pd*** et dans le fichier ***8.recursion.pd*** de Nicolas.

On peut également faire une double itération qui permet par exemple de créer des tableaux (grilles à deux dimensions). Voir fichier ***8.B.double_recursion.pd***.

Vendredi 6 août 2010 PM

On a déjà vu ***iAmbiant2D*** qui permettait de créer un champs de force de type gravité. Le « ***i*** » qui commence le nom de ces objets est l'initiale de « **i**nteracteur ». Ces interacteurs vont également nous permettre de créer des chocs entre les masses. Il suffit de mettre des iCircle2D qui simulent des champs de forces répulsionnels quand ils sont appliqués à chacune des masses. L'exemple que l'on retrouve dans Browser -> ***examples/pmpd/24_sand.pd***.

Pour créer une multitude d'objets rapidement sans faire de copier/coller il est possible de les créer « à la volée ». Pour cela on envoi des messages ***obj*** à Pure Data directement pour lui demander de créer des objets.

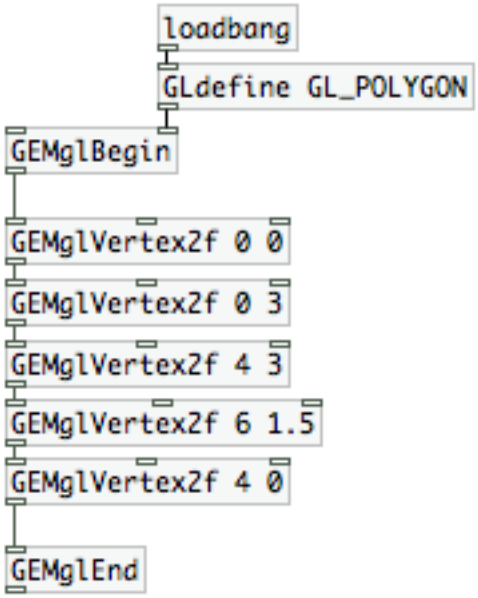
On peut dessiner en openGL directement dans Pure Data. On trouve dans Browser -> ***examples/Gem/openGL/01.primQuad.pd*** un bon exemple pour démarrer. Premièrement, on doit savoir que toutes les commandes *openGL* (que l'on trouve dans la documentation *openGL*) sont utilisables dans Pure Data en utilisant des objets dont le nom commence par ***GEM***. Par exemple, la fonction *glRotatef* d'*openGL* peut être utilisé dans l'objet ***GEMglRotatef***. Par contre, quand on veut utiliser les constante *openGL* telle que *GL_RECTANGLE*, par exemple, on doit systématiquement utiliser un objet ***GLdefine*** suivi du nom de la constante.

Pour illustrer ceci, on peut regarder attentivement ce morceau de code *openGL*, à gauche, qui est « traduit » en Pure Data à droite:

```
glBegin(GL_POLYGON);


   glVertex2f(0.0, 0.0);
   glVertex2f(0.0, 3.0);
   glVertex2f(4.0, 3.0);
   glVertex2f(6.0, 1.5);
   glVertex2f(4.0, 0.0);

glEnd();
```



On va maintenant essayer de réaliser un graphique de type « répartition des élus à l'assemblée nationale » (un demi camembert). La solution se trouve dans ***Exercice23.pd***.

Attention: ne pas oublier l'utilité de l'objet ***gemlist*** qui stocke une « valeur » de ***gemhead*** à un instant donné.

Le shader (GLSL pour Graphic Library Shading Laguage) est complémentaire de l'*openGL* mais n'est pas compatible avec toutes les cartes graphiques. Il faut donc vérifier, par exemple, avec le premier exemple que l'on retrouve dans le Browser -> ***examples/Gem/glsl/01.simple_texture.pd***.

Le ***gemframebuffer*** est un objet qui permet de « capturer » un ***gemhead*** dans la chaîne pour la réutiliser dans une autre chaîne comme texture, par exemple. Voir le fichier du Browser -> ***examples/Gem/texture/10.framebuffer.pd***.