43-Conception d'interfaces utilisateur en JavaScript

introduction

L'objet *jsui* vous permet d'utiliser JavaScript pour concevoir des objets d'interface utilisateur graphique à employer dans l'environnement Max. L'implémentation JavaScript de l'objet *jsui* est similaire à celle utilisée dans l'objet *js*, avec une API supplémentaire qui prend en charge les graphiques vectoriels à deux et trois dimensions dessinés avec des commandes OpenGL. Elle comprend également des méthodes pour gérer l'interaction avec la souris dans la fenêtre de l'objet *jsui*.

En plus des avantages fournis par JavaScript, *jsui* fournit un certain nombre de fonctionnalités intégrées qui rendent le développement de l'interface utilisateur flexible:

- Les objets *jsui* dessinent leurs géométries par rapport à la taille de la boîte de l'objet *jsui*; le redimensionnement d'un objet *jsui* redimensionnera correctement tous les éléments dessinés à l'intérieur de celui-ci.
- Les objets *jsui* fonctionnent avec un langage graphique vectoriel (OpenGL) qui supporte une grande variété de formes simples et de primitives de dessin. En outre, un certain nombre de fonctions graphiques de plus haut niveau sont disponibles. L'objet *jsui* peut également effectuer un anti-aliasing sur l'image pour vous donner un objet aussi lisse que possible, bien que cela s'accompagne d'une diminution des performances.
- L'objet *jsui* vous permet de dessiner une scène qui dépasse les limites de la boîte d'objet. En ajustant l'orientation de la caméra dans l'espace OpenGL, vous pouvez créer et gérer différentes "vues" du même objet d'interface utilisateur. En plus d'utiliser l'objet *jsui* pour la conception d'interfaces utilisateur, on peut utiliser l'objet simplement comme moteur graphique OpenGL intégré au patcheur Max pour les opérations de dessin algorithmiques.

Ce didacticiel suppose que vous avez déjà consulté les autres didacticiels JavaScript. L'objet *jsui* base la plupart de son langage graphique sur des fonctions OpenGL, dont les spécificités dépassent le cadre de ce tutoriel. OpenGL 'Redbook' est la référence standard pour ces fonctions. Une version en ligne est disponible à l'adresse suivante: http://www.opengl.org/documentation/red_book/

L'API OpenGL supportée par *jsui* est contenue dans un objet appelé *jsui* **sketch**. Cet objet comprend la plupart des commandes OpenGL et des constantes symboliques. La conversion entre du code OpenGL (par exemple, tel qu'il est donné en C dans les exemples de code du «Redbook») et les méthodes et propriétés de **sketch** pour le code JavaScript *jsui* est relativement simple si vous observez les instructions suivantes:

- Toutes les commandes OpenGL sont en minuscule dans l'objet *jsui* **sketch**, par exemple. **glColor ()** devient sketch.**glcolor ()**.
- Les constantes symboliques OpenGL, en plus des minuscules, perdent leur préfixe «GL_», de sorte que **GL_CLIP_PLANE1** devient **clip_ plane1 ()**, par exemple.

Un certain nombre de commandes de dessin et de forme de plus haut niveau sont disponibles, et peuvent accélérer le développement de l'interface utilisateur. La référence de *jsui* sketch contenue dans le guide *Javascript in Max* (et le patch d'aide pour l'objet *jsui*) contient les listes de ces commandes.

Pour ouvrir le patch du didacticiel, cliquez sur le bouton vert **Ouvrir didacticiel** dans le coin supérieur droit de la fenêtre de documentation.

jsui en action

Jetez un coup d'oeil à notre patcheur de tutoriel. Vous verrez un objet *jsui* contenant une grille de cercles rouge clair sur un fond vert. En cliquant sur un cercle à l'intérieur de l'objet, la couleur du cercle devient un rouge plus foncé. En cliquant à nouveau sur le même cercle, la couleur redeviendra rouge clair.

Cliquez sur certains cercles pour les mettre en surbrillance (en rouge foncé). À la droite du patch, manipulez certains des objets *sliders* situés au-dessus de l'objet *router*. Notez la correspondance entre les cercles sur lesquels vous cliquez et les objets *sliders* situés sous le *router* qui reprennent les valeurs du dessus. Cliquez sur la boîte de *message* étiquetée clear attachée à l'objet *jsui*. Les cercles doivent tous passer au rouge clair et l'objet *router* ne transmettra plus les messages des objets *sliders* situés au-dessus. Cliquez sur la boîte de *message* contenant la liste **0 0 1, 1 1 1**, etc. L'objet *jsui* devrait se mettre à jour pour afficher une rangée en diagonale. L'objet *router* transmettra maintenant les messages du premier curseur au-dessus au premier curseur en dessous, et ainsi de suite.

Notre objet *jsui* utilise le code JavaScript pour émuler certaines des fonctionnalités de l'objet Max *matrixctrl*. Les colonnes représentent l'entrée de l'objet *router ;* les lignes spécifient la sortie. Notre objet *jsui* communique avec l'objet *router* en envoyant des listes (au format input output state) qui indiquent à l'objet *router* de transmettre les messages reçus à une entrée à toutes les sorties appropriées, compte tenu de la configuration actuelle de l'objet *jsui*.

Comme pour l'objet *js*, l'objet *jsui* obtient son programme d'un fichier écrit en JavaScript, enregistré quelque part dans le chemin de recherche. Comme il s'agit d'un objet graphique, il n'y a pas de boîte d'objet pour saisir le nom du fichier. Au lieu de cela, nous définissons le fichier source JavaScript en utilisant **l'inspecteur** de l'objet *jsui*.

Déverrouillez le patch du tutoriel et mettez en surbrillance l'objet *jsui*. Dans le menu **Objet**, sélectionnez **Lire les informations**…. Un inspecteur doit apparaître avec le nom de notre fichier source *jsui* ("mymatrix.js") dans le champ de texte intitulé "JavaScript File".

Vous pouvez également définir la taille de l'objet dans l'inspecteur, ainsi qu'activer ou désactiver une bordure autour de l'objet. La désactivation de la bordure de l'objet, associée au réglage de la couleur de fond de votre patch Max pour qu'elle corresponde à celle de votre objet *jsui*, peut vous aider à concevoir une interface utilisateur transparente.

Le code de dessin

L'objet *jsui* est graphique, un double-clic sur l'objet n'ouvrira pas l'éditeur de texte comme avec l'objet *js*. Au lieu de cela, cliquez sur la boîte de message intitulée **Open**. L'éditeur de texte contenant notre fichier JavaScript («mymatrix.js») apparaîtra. Notre fichier JavaScript est enregistré sur le disque dans le même dossier que le patch du didacticiel.

Comme avec un script *js*, notre code pour l'objet *jsui* commence par un bloc global qui nous permet de définir des entrées et des sorties pour l'objet et des variables globales pour notre code. C'est également là que nous tapons les commandes que nous voulons voir se produire lorsque l'objet est initialisé:

```
// inlets and outlets
inlets = 1;
outlets = 1;
```

```
// global variables
var ncols=4; // default columns
var nrows=4; // default rows
var vbrgb = [0.8,1.,0.8,0.5];
var vmrgb = [0.9,0.5,0.5,0.75];
var vfrgb = [1.,0.,0.2,1.];
// initialize state array
var state = new Array(8);
for(var i=0;i<8;i++)
{
        state[i] = new Array(8);
        for(j=0;j<64;j++)
        {
                state[i][j] = 0;
        }
}
// set up jsui defaults to 2d
sketch.default2d();
// initialize graphics
draw();
refresh();
```

Notre code JavaScript définit deux variables globales (**ncols** et **nrows**) accessibles à toutes les fonctions, ainsi qu'un certain nombre d'objets **Array** globaux définissant les couleurs du dessin et un tableau d'état que nous utiliserons pour stocker des informations sur les cercles qui sont 'on' et qui sont 'off' dans notre interface utilisateur.

Les tableaux multidimensionnels en JavaScript sont alloués par un objet **Array** dans lequel chaque élément du tableau est lui-même un objet Array (et ainsi de suite, si plus de deux dimensions sont nécessaires). Cela peut sembler quelque peu inhabituel si vous avez travaillé dans d'autres langages de programmation où les tableaux multidimensionnels peuvent être déclarés directement (par exemple en C). Les boucles **for ()** de notre bloc global réalisent cette allocation et initialisent tous les éléments du tableau d'état à **0**. Une fois qu'un tableau multidimensionnel est créé, il peut être référencé en utilisant la notation courante entre crochets, par exemple state [4] [2].

Après nos déclarations de variables et de tableaux, nous trouvons trois commandes qui font référence au comportement graphique de l'objet *jsui*. Le premier, **sketch.default2d ()**, demande à notre objet *jsui* d'initialiser un certain nombre de comportements par défaut en partant du principe que nous lui donnerons des commandes graphiques pour une scène à deux dimensions. Elle définit une vue par défaut sur le contexte de rendu OpenGL et exécute un certain nombre de routines utilitaires pour nous permettre de commencer à placer des éléments graphiques dans la fenêtre. La commande **draw ()** (qui pourrait être nommée n'importe comment) fait référence à notre fonction graphique principale que nous écrivons pour contenir toutes les commandes nécessaires pour dessiner l'interface utilisateur de l'objet *jsui*. La commande **refresh ()** copie le backbuffer OpenGL (où le dessin est fait initialement pour éviter le scintillement) sur l'écran actuel. Mettre un commentaire sur la commande **refresh ()** empêchera notre objet *jsui* de nous montrer quoi que ce soit.

En dessous du bloc global, examinez la fonction **draw ()**. C'est la fonction qui fournit à *jsui* toutes les commandes nécessaires pour dessiner notre interface d'écran:

```
function draw()
{
	with (sketch)
	{
		// set how the polygons are rendered
		glclearcolor(vbrgb[0],vbrgb[1],vbrgb[2],vbrgb[3]); // set the clear color
		glclear(); // erase the background
		var colstep=2./ncols; // how much to move over per column
		var rowstep=2./nrows; // how much to move over per row
		for(var i=0;i<ncols;i++) // iterate through the columns
		{
			for(var j=0;j<nrows;j++) // iterate through the rows
			{
				moveto((i*colstep + colstep/2)-1.0, 1.0 - (j*rowstep + rowstep/2),
				0.); // move the drawing point
				if(state[i][j]) // set 'on' color
				{
					glcolor(vfrgb[0],vfrgb[1],vfrgb[2],vfrgb[3]);
				}
				else // set 'off' color (midway between vbrgb and vfrgb)
				{
					glcolor(vmrgb[0],vmrgb[1],vmrgb[2],vmrgb[3]);
				}
				circle(0.7/Math.max(nrows,ncols)); // draw the circle
			}
		}
	}
}
```

Les commandes graphiques (tout ce qui commence par «gl», ainsi que la commande **circle ()**) sont toutes des méthodes et des propriétés de l'objet **sketch**, qui encapsule la plupart des API OpenGL, tout comme l'objet **Math** encapsule les fonctions mathématiques les plus courantes.

L'instruction JavaScript **with ()** nous permet d'utiliser des propriétés et des méthodes appartenant à un objet (dans ce cas, l'objet **sketch** qui nous fournit la fonctionnalité OpenGL) sans devoir faire référence à 'sketch' dans chaque commande. Sans l'instruction **with ()**, nous devrions écrire **sketch.glcolor ()** au lieu de **glcolor ()**, **sketch.circle ()** au lieu de **circle ()**, etc. Cette astuce utile pourrait également être mise en œuvre dans des fonctions qui dépendent fortement d'autres objets (par exemple, **Task** ou **Patcher**).

Notre fonction **draw ()** définit certains comportements de dessin par défaut et efface la fenêtre avec la couleur définie par le tableau **vbrgb**. Elle itère ensuite dans notre tableau d'états en fonction du nombre de colonnes (**ncols**) et de rangées (**nrows**) que nous avons défini pour notre objet. Si un élément d'état particulier est **0** (off), il dessine un cercle dans la couleur définie par **vmrgb**. Si un élément est à **1** (on), le cercle est dessiné dans la couleur **vfrgb**. La position des cercles est déterminée par le nombre de rangées et de colonnes et est basée sur les limites de notre monde OpenGL, qui doivent être comprises entre **-1,0** et **1,0** sur les deux axes (v'est à dire que le milieu de notre fenêtre *jsui* en coordonnées OpenGL est **0**, **0**).

La taille de l'univers n'est pas limitée à des coordonnées comprises entre **-1,0** et **1,0**; notre fenêtre d'affichage par défaut nous place simplement au centre de la scène dont la plage **y** est comprise entre **-1,0** et **1,0** et dont la plage **x** est mise à l'échelle en fonction du rapport d'aspect de l'objet.

Comme notre boîte à objets est carrée, nos plages sont les mêmes sur les deux axes. En changeant la fenêtre d'affichage (en manipulant la position de notre «caméra» virtuelle, par exemple) les coordonnées visibles dans la boîte de l'objet *jsui* seront modifiées.

Réglage des paramètres

Dans le patch du didacticiel, modifiez l'objet boîte de *nombre* qui définit les rangées et les colonnes. Notez que l'objet ajoutera dynamiquement jusqu'à huit rangées et colonnes de cercles en fonction de ces valeurs. Regardez les fonctions **rows ()** et **cols ()** dans le code JavaScript. Notez qu'ils appellent une fonction **bang ()** après avoir défini leurs variables.

```
// rows -- change number of rows in jsui
function rows(val)
{
	if(arguments.length)
	{
		nrows=arguments[0];
		bang(); // draw and refresh display
	}
}
// cols -- change number of columns is jsui
function cols(val)
{
	if(arguments.length)
	{
		ncols=arguments[0];
		bang(); // draw and refresh display
	}
}
```

La fonction **bang ()**, que nous appelons après presque tous les changements apportés à l'objet par Max (y compris les événements de souris), appelle simplement **draw ()** et **refresh ()** comme nous l'avons fait dans notre bloc global, ce qui amène l'objet *jsui* à mettre à jour sa fenêtre. pour refléter les changements graphiques.

```
/ bang -- draw and refresh display
function bang()
{
	draw();
	refresh();
}
```

En n'effectuant le dessin que lorsque cela est nécessaire, nous sommes en mesure de réduire le temps processeur utilisée par l'objet.

Dans le patcheur du didacticiel, modifiez les objets *swatch* qui définissent les messages **frgb** et **brgb** en notre objet *jsui*. Regardez les fonctions correspondantes (**frgb ()** et **brgb ()**) dans le code JavaScript. Notez que le tableau pour la couleur du cercle "off" (**vmrgb**) est situé à mi-chemin entre les couleurs définies par les messages frgb et brgb:

```
frgb -- change foreground (clicked) sphere color
function frgb(r,g,b)
{
	vfrgb[0] = r/255.;
	vfrgb[1] = g/255.;
	vfrgb[2] = b/255.;
	vmrgb[0] = 0.5*(vfrgb[0]+vbrgb[0]);
	vmrgb[1] = 0.5*(vfrgb[1]+vbrgb[1]);
	vmrgb[2] = 0.5*(vfrgb[2]+vbrgb[2]);
	bang(); // draw and refresh display
}

// brgb -- change background color
function brgb(r,g,b)
{
	vbrgb[0] = r/255.;
	vbrgb[1] = g/255.;
	vbrgb[2] = b/255.;
	vmrgb[0] = 0.5*(vfrgb[0]+vbrgb[0]);
	vmrgb[1] = 0.5*(vfrgb[1]+vbrgb[1]);
	vmrgb[2] = 0.5*(vfrgb[2]+vbrgb[2]);
	bang(); // draw and refresh display
}
```

La couleur dans OpenGL est représentée par quatre valeurs en virgule flottante dans la plage **0.0-1.0**, correspondant respectivement aux rouge, vert, bleu et alpha (transparence). Cela contraste avec de nombreux systèmes vidéo qui font généralement référence à la couleur dans le nombre entier compris entre **0** et **255** (sans valeur alpha). La plupart du travail dans nos fonctions **frgb ()** et **brgb ()** consiste à convertir la dernière (utilisée par l'objet *swatch*) en la première (comprise par l'objet *jsui*).

Interaction de la souris

Déverrouillez le patcheur du tutoriel et redimensionnez l'objet *jsui* (les cercles doivent être redimensionnés de manière dynamique). Verrouillez le patcheur et remarquez que les clics de la souris modifient toujours les états des cercles corrects. Verrouillez à nouveau le patcheur et regardez la fonction **onclick ()** dans le code JavaScript.

Les fonctions **onclick ()**, **ondblclick ()** et **ondrag ()**, une fois définies, indiquent à notre objet *jsui* ce qu'il doit faire lorsqu'un utilisateur clique, double-clique ou fait glisser la souris sur l'objet. La fonction est appelée avec des arguments indiquant l'endroit de la fenêtre de l'objet où l'action s'est produite, ainsi qu'un certain nombre d'indicateurs (comme le fait que la souris ait été enfoncée, l'état de la touche shift, etc.). Dans notre fonction onclick (), nous n'utilisons que les deux premiers arguments, correspondant au **x** et au **y** du clic de la souris.

```
// onclick -- deal with mouse click event
function onclick(x,y)
{
	var worldx = sketch.screentoworld(x,y)[0];
	var worldy = sketch.screentoworld(x,y)[1];
	var colwidth = 2./ncols; // width of a column, in world coordinates
	var rowheight = 2./nrows; // width of a row, in world coordinates
```

```
    var x_click = Math.floor((worldx+1.)/colwidth); // which column we clicked
    var y_click = Math.floor((1.-worldy)/rowheight); // which row we clicked
    state[x_click][y_click] = !state[x_click][y_click]; // flip the state of the clicked point
    outlet(0, x_click, y_click, state[x_click][y_click]); // output the coordinates and state of the
    clicked point
    bang(); // draw and refresh display
}
```

Notre univers graphique OpenGL est défini en termes de coordonnées à virgule flottante (dans notre cas, entre **-1,0** et **1,0**). Les fonctions de la souris *jsui* renvoient des coordonnées basées sur le ***pixel*** (en partant du coin supérieur gauche de la boîte de l'objet) où l'événement souris s'est produit. Nous devons convertir ces deux systèmes (de coordonnées universelles et de coordonnées d'écran, respectivement) afin d'évaluer correctement les événements de souris pour notre grille de cercles. Les méthodes sketch **worldtoscreen ()** et **screentoworld ()** effectuent ces conversions pour nous:

```
var worldx = sketch.screentoworld(x,y)[0];
var worldy = sketch.screentoworld(x,y)[1];
```

Une fois que nous connaissons la largeur et la hauteur de l'endroit où nous avons cliqué, nous pouvons le subdiviser en fonction du nombre de cercles que nous avons sur chaque axe:

```
var colwidth = 2./ncols; // width of a column, in world coordinates
var rowheight = 2./nrows; // width of a row, in world coordinates
```

Nous pouvons ensuite introduire les coordonnées du clic de la souris pour déterminer le cercle le plus proche sur lequel nous avons cliqué:

```
x_click = Math.floor((worldx + 1) / colwidth) // colonne sur laquelle nous avons cliqué
y_click = Math.floor((1 - worldy) / rowheight) // la ligne sur laquelle nous avons cliqué.
```

Nous inversons ensuite l'élément du tableau state correspondant au cercle sur lequel nous avons cliqué :

```
state[x_click][y_click] = !state[x_click][y_click]; // flip the state of the clicked point
```

Après avoir défini correctement le tableau d'états, nous envoyons une liste correspondant au changement de la sortie de notre objet *jsui* dans Max. Nous produisons ensuite un **bang** sur notre propre objet *jsui*, en mettant à jour les graphiques pour refléter le changement:

```
outlet(0, x_click, y_click, state[x_click][y_click]); // output the coordinates and state of the clicked
point
bang(); // draw and refresh display
```

Notez que nous avons défini la fonction **onclick ()** comme étant locale, de sorte qu'elle ne puisse pas être déclenchée à partir d'un message **onclick** envoyé par notre patch Max.

Familiarisez-vous avec le code JavaScript et la façon dont il est lié au comportement de l'objet *jsui* dans le patcheur. Cliquez sur l'objet *toggle* pour activer l'objet *metro* à la droite du patch. Cela simulera une certaine entrée de la part des objets *slider*. Placez les instructions **post ()** dans le code JavaScript pour faciliter la navigation dans les valeurs lorsqu'elles passent du clic de la souris à la sortie de la liste.

Résumé

L'objet *jsui* est un outil puissant qui vous permet de concevoir et de mettre en oeuvre des interfaces utilisateur personnalisables en utilisant JavaScript comme langage de programmation. Les points clés du programme concernent la fonction principale de dessin (qui définit une séquence de commandes pour décrire l'affichage graphique de l'objet *jsui*) et les fonctions d'interaction avec la souris **onclick ()**, **ondblclick ()** et **ondrag ()**. Il est important de noter les différences de représentation des couleurs (virgule flottante ou nombre entier) et des coordonnées spatiales (coordonnées universelles en virgule flottante ou coordonnées cartésiennes en pixels) entre l'API OpenGL utilisée dans l'objet *jsui* et l'environnement Max, respectivement.