EXTENSION DE GUIDO AUX NOTATIONS CONTEMPORAINES

Camille Le Roi, Colas Decron, Dominique Fober
Grame - Centre national de création musicale
{cleroi, decron, fober}@grame.fr

RÉSUMÉ

Le projet GUIDO regroupe à la fois un format textuel de description de partitions musicales, un moteur de rendu basé sur ce format, et une librairie qui fournit aux développeurs d'applications un support de haut niveau pour l'ensemble des services liés au format GUIDO et à son rendu sous forme graphique. A l'origine, le projet GUIDO traite de la notation traditionnelle de la musique occidentale. Le projet a été récemment étendu, tant du point de vue du format que du point de vue du rendu, pour adresser également les besoins les plus courants de la musique contemporaine. Cet article présente ces extensions et montre comment les choix de design fournissent à l'utilisateur une large palette de solutions pour la notation de la musique.

1. INTRODUCTION

Le projet GUIDO est né il y a presque 20 ans, tout d'abord comme format textuel de représentation de la musique [7, 8], puis comme moteur de rendu graphique de partitions musicales [14]. Lors de sa conception, le format GUIDO se différencie des approches proposées par SCORE [10], MuseData [5], DARMS [15] ou encore Humdrum [9] notamment parce qu'il est pleinement lisible par l'utilisateur. L'approche développée par GUIDO repose sur l'idée d'adéquation, qui invite à noter simplement les concepts musicaux simples, et à ne recourir à une syntaxe complexe que pour les notions musicales complexes.

Postérieurement au format GUIDO, d'autres représentations textuelles de la musique ont vu le jour, telles que Lilypond [12, 11] qui est assez similaire dans son format, ou MusicXML [3], orientée vers l'échange de partitions. Toutefois, le projet GUIDO reste unique car il est le seul qui propose à la fois une syntaxe, un moteur de rendu et une librairie C/C++ permettant d'embarquer des capacités de rendu de partitions dans des applications indépendantes. Par ailleurs, la rapidité et l'efficacité du moteur de rendu le rendent utilisable dans un contexte *temps réel* (pour des partitions dont le niveau de complexité n'est pas trop élevé), ouvrant la voie à des formes d'interactivité inédites [6, 2].

Pour répondre aux besoins les plus courants de la musique contemporaine, le format GUIDO ainsi que le moteur de rendu ont été étendus. Après un bref rappel sur le format GUIDO, cet article décrit ces extensions (nouveaux symboles puis améliorations des symboles existants), et situe les choix faits en matière de rendu graphique dans les pratiques identifiées dans la littérature contemporaine. Nous montrerons également comment une forme de *langage générique* permet de combiner ces représentations et offre à l'utilisateur un large éventail de solutions pour la représentation de la musique.

2. LE FORMAT DE NOTATION GUIDO

Ci-dessous figurent les principes généraux du format de notation GUIDO. Ils sont fournis pour permettre la compréhension du contenu des sections suivantes. Pour plus de détails on peut se reporter à [1] ou [8].

Le format de base de GUIDO recouvre les notes, silences, altérations, voix indépendantes, et les concepts les plus courants de la notation musicale comme les clefs, métriques, armures, articulations, liaisons, etc. Les notes sont représentées par leur nom `(a b c d e f g h)`, une altération optionnelle (`#` et `&` pour dièse et bémol), un numéro d'octave optionnel (`1` par défaut) et une durée optionnelle exprimée sous forme de fraction (`1/4` pour la noire, `1/8` pour la croche, `1/2` pour la ronde, etc.).

Les *tags* sont utilisés pour donner des informations musicales supplémentaires, comme les liaisons, clefs, armures, etc. Un *tag* a une des formes suivantes :

```
\tagname<param-list>
\tagname<param-list>(note-series)
```

où *param-list* est une liste optionnelle de paramètres (chaînes de caractères ou nombres), et où *note-series* est l'ensemble des notes ou accords concernés par le *tag*. Dans ce cas nous parlerons de *range-tag* (*tag* possédant une durée explicite).

3. EXTENSIONS DE LA NOTATION

3.1. Micro-tonalité

La micro-tonalité est limitée graphiquement à la représentation du quart de ton, alors que, du point de vue du langage, elle s'exprime comme un nombre flottant de demi-tons qui sont arrondis au quart de ton le plus proche pour le rendu graphique :

```
1)  \alter<detune>
2)  \alter<detune>(note-series)
```

La première forme (1) introduit la notion d'*altération courante* : l'altération reste valide jusqu'à une indication contraire. Dans la deuxième forme (2), l'altération ne s'applique qu'aux notes indiquées.

EXEMPLE

```
[
  a&& \alter<-1.5>(a) a& \alter<-0.5>(a)
  a \alter<0.5>(a) a# \alter<1.5>(a)
  \alter<1.8>(a)
]
```

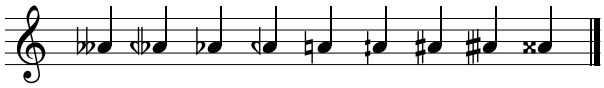


Figure 1. Micro-tonalité arrondie au quart de ton

La micro-tonalité est également introduite dans les armures de clef libres. De manière similaire, elle s'exprime sous forme de nombres flottants indiqués entre crochets.

EXEMPLE

```
[
  \key<"free=c[1.5]g#b&a[-1.5]"> a g#
  \alter<0.5> g  a \alter<1.5> g
]
```

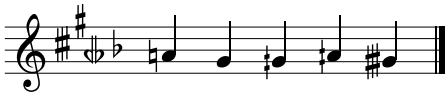


Figure 2. Armure de clef libre avec micro-tonalité

3.2. Têtes de notes

La librairie GUIDO a été étendue avec 6 nouveaux symboles de têtes de notes. Son implémentation repose sur le tag `\noteFormat`, déjà existant, qui modifie l'apparence des notes qui le suivent (couleur, taille, offset, etc.), et auquel nous avons ajouté un attribut de style, permettant à l'utilisateur de choisir l'apparence des têtes de notes.

```
\noteFormat<style=noteHeadStyle>
```

où `noteHeadStyle` est parmi :

- `x` : une croix
- `diamond` : un losange
- `round` : un rond
- `square` : un carré
- `triangle` : un triangle, pointe en haut
- `reversedTriangle` : un triangle, pointe en bas

La première portée de l'exemple de la figure 3 est générée avec le code suivant :

```
[
  \noteFormat<style="x"> a
  \noteFormat<style="diamond"> a
  \noteFormat<style="round"> a
  \noteFormat<style="square"> a
  \noteFormat<style="triangle"> a
  \noteFormat<style="reversedTriangle"> a
]
```

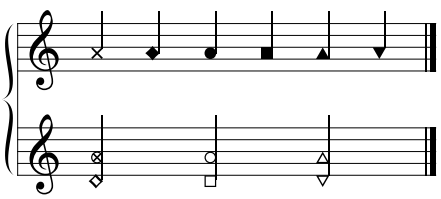


Figure 3. Les 6 nouveaux *noteheads* ajoutés

Pour l'apparence graphique de ces éléments, nous nous sommes principalement inspirés des ouvrages de E. Gould [4] et de K. Stone [16].

Des ajustements particuliers ont été nécessaires pour tenir compte des différences graphiques entre les têtes de notes :

- insertion d'un offset horizontal variable selon le symbole choisi, l'orientation de la hampe, l'utilisation en accord ou non (les accords pouvant posséder des *noteheads* différents pour chacune de leurs notes) ;
- adaptation de la longueur de la hampe selon le type de symbole utilisé (par exemple, différence de longueur entre la croix et le losange).

3.3. Métriques complexes

La métrique complexe permet d'indiquer une subdivision de la métrique par une somme en place et lieu de la métrique habituelle. La syntaxe est la même que pour une métrique normale, elle fait usage du tag `\meter` et il suffit de remplacer le numérateur par une somme, comme le montre l'exemple ci-dessous, illustré en figure 4.

EXEMPLE

```
[ \meter<"3+3+2/4"> a b g ]
```

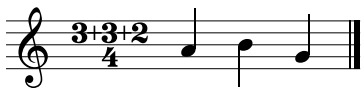


Figure 4. Métrique complexe

3.4. *Clusters*

Un *cluster* est un accord musical contenant plusieurs notes consécutives. Il peut par exemple être effectué sur un piano où il se joue en déclenchant des touches contiguës (en utilisant son avant-bras, par exemple).

Un *cluster* est indiqué par un range-tag s'appliquant à des accords de deux notes : les notes extrêmes du *cluster* (les autres notes éventuelles seront ignorées).

```
\cluster<param-list>(chord-series)
```

Comme l'indique en partie la figure 5, les choix, faits à partir d'ouvrages référents [4, 16], ont été les suivants :

- le *cluster* est représenté par un rectangle ;

EXEMPLE

```
[
  \cluster({a, d} {c/8, f}
      {a/2, d2} {c1/1, f2})
]
```

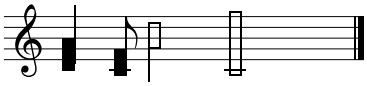


Figure 5. *Cluster*

– ce rectangle est plein ou vide, selon la durée du *cluster* ;
– la hampe est conservée pour garder l'indication de la durée.

Le tag `\cluster` supporte également les paramètres de contrôle standards (`size`, `dx`, `dy`, `color`) auxquels ont été ajoutés des paramètres `hdx` et `hdy` permettant d'introduire un décalage s'appliquant à la tête de *cluster*.

Un *cluster* est vu comme une note et supporte donc le tag `\noteFormat` ainsi que le contrôle de l'orientation des têtes de notes (`\headsReverse`, `\headsLeft` et `\headsRight`) ainsi qu'illustré en figure 6.

EXEMPLE

```
[
  \cluster<color="red", hdx=1, hdy=3>({a})
  \cluster<size=0.5>({f, c2})
  \noteFormat<color="purple">
  \headsReverse
  \cluster<color="green", size=2>({f, g2})
  \cluster<"blue">({d1/2, g})
]
```

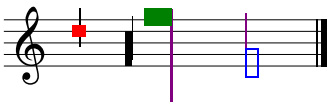


Figure 6. *Clusters* : interactions avec d'autres tags

3.5. *Glissando*

Le *glissando* se présente classiquement comme une ligne joignant deux notes décalées dans le temps (figure 7), indiquant ainsi un glissement, continu ou non suivant le type d'instrument, d'une hauteur à une autre.

La syntaxe choisie a été la suivante :

```
\glissando<params>(notes)
params :
  - fill = "true" ou "false" :
    option de remplissage
  - thickness : épaisseur de la ligne
```

Il paraissait important de donner la possibilité à l'utilisateur de n'écrire qu'une seule fois le tag pour décrire plusieurs *glissandi* consécutifs, contrairement à la description de lilypond où le tag doit être répété entre chaque note. Ainsi le range-tag apparaissait comme la meilleure solution pour décrire un ou plusieurs *glissandi* : toutes les notes comprises dans les parenthèses seraient prises en compte et liées les unes aux autres par des *glissandi*.

Concernant le moteur de rendu, l'algorithme de démultiplication d'un même tag et de répartition de celui-ci est fortement inspiré de celui du `\tie`, ou liaison de prolongation, qui doit également créer une liaison entre chacune des notes affectées.

EXEMPLE

```
[ \glissando(g b d) ]
```

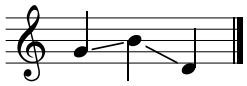


Figure 7. *Glissando*

3.6. Liens de croches en soufflet (*feathered beaming*)

Décrite par Gardner Read comme "a highly graphic representation of rythmic flexibility" [13] (p. 94), cette notation contemporaine d'un *accelerando* ou *ritardando* à travers les liens de croches est, de manière générale, encore peu répandue dans les ouvrages de théorie, et se résume souvent à des cas simples, partant ou arrivant à la simple croche, c'est à dire partant de, ou arrivant vers, un unique point. Pourtant il paraît intéressant de pouvoir offrir la possibilité aux compositeurs de décrire le passage entre n'importe quelles valeurs : par exemple d'une double-croche à une triple-croche, ou d'une quadruple-croche à une double-croche, etc. (figure 8).

EXEMPLE

```
[
 \fBeam(a/16 a a a/32) \fBeam(a/64 a a a a/16)
]
```

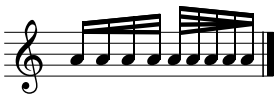


Figure 8. *Feathered beams* plus complexes

Dans un désir de simplicité, il a été décidé de décrire par un seul range-tag un seul groupe de notes liées, correspondant à un point de départ et un point d'arrivée uniques.

Il est cependant possible de chaîner plusieurs *feathered beams* en utilisant la forme `Begin / End` des range-tags

(i.e \fBeamBegin et \fBeamEnd), permettant alors, non seulement de chaîner deux *beams* par une note (figure 9), mais aussi de les faire se chevaucher sur plusieurs notes.

EXEMPLE

```
[
  \fBeamBegin:1 c/8 d e/16
  \fBeamBegin:2 f/32 \fBeamEnd:1
  e/16 d/8 c \fBeamEnd:2
]
```

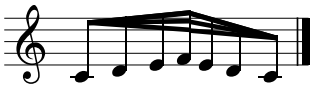


Figure 9. *Feathered beams* chaînés

On remarque que, même dans un ouvrage tel que celui d'E. Gould [4], peu de détails sont donnés sur cette notation, et on se heurte rapidement à des incohérences telles que celle de la figure 10 tirée de ce même ouvrage : on veut faire tenir en un temps une croche et quatre notes de durées inférieures à la croche mais supérieures à la triple-croche.

Figure 10. Exemple d'incohérence entre la durée totale et les durées individuelles des notes d'un groupe lié (Behind Bars, p. 158).

Ce manque d'une exacte cohérence entre tempo et aspect graphique est également souligné par Kurt Stone [16] : "Besides, the gradual increase or decrease in the number of beams makes exact indications of beat-units impossible" (p. 124).

Ainsi on voit l'importance des choix de design de ce symbole, sur la manière de le décrire textuellement, de le dessiner en particulier dans le cas de plusieurs voix simultanées, et de manière générale sur la liberté laissée à l'utilisateur.

Concernant les possibilités de description données à l'utilisateur, nous avons pu voir que plusieurs critères, parfois incohérents entre eux, pouvaient intervenir dans la description et devaient pouvoir être imposés par l'utilisateur :

- le nombre de notes dans le groupe ;
- leurs durées individuelles et intrinsèques qui définissent également leur espacement ;
- la durée totale du groupe, directement dépendante des durées individuelles ;
- les durées de départ et d'arrivée qui définissent l'aspect graphique du *feathered beam*.

Ces différents niveaux de description nous poussent à faire une claire distinction entre l'aspect temporel et l'aspect graphique de notre groupe de notes. Il a été décidé de laisser au compositeur la liberté de donner à son *feathered beam* l'aspect graphique qu'il désire, indépendamment des durées internes de ses notes. Il pourra ainsi jouer sur les espacements entre notes, sur le nombre de notes, et sur la durée totale de manière "classique" en imposant à chaque note une durée, et en veillant à ce que la somme de toutes les notes corresponde à la durée totale souhaitée. Cette durée totale peut être indiquée sur la partition grâce au paramètre drawDuration (figure 11).

EXEMPLE

```
[ \fBeam<drawDuration="true"> (
    a/8 a a/16 a a a/32 a )]
```

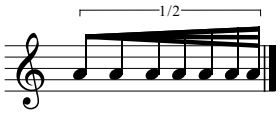


Figure 11. Représentation de la durée

Sans plus d'indication de sa part, l'aspect graphique suivra les durées réelles des notes et prendra comme points de départ et d'arrivée les durées des première et dernière notes. C'est le cas de la figure 11.

En revanche, le *param* durations pourra lui permettre d'imposer un aspect graphique tout autre, lui donnant ainsi la possibilité d'indiquer à l'interprète un changement de tempo plus ou moins important, et pas nécessairement cohérent avec les durées intrinsèques des notes, mais ayant du sens dans l'interprétation (figure 12).

EXEMPLE

```
[ \fBeam<durations="1/16, 1/64",
      drawDuration="true">(
    a/8 a/16 a a a/32 a )]
```

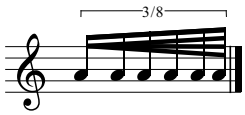


Figure 12. Aspect graphique imposé par l'utilisateur

En résumé, la syntaxe générique est la suivante :

```
\fBeam<params>(notes)
params :
  - durations = "firstDur", "lastDur" :
    durée des première et dernière notes
  - drawDuration :
    indication de la durée totale
```

3.7. Tags *\staffOff* et *\staffOn*

Les tags \staffOff et \staffOn permettent de faire disparaître et apparaître des parties de la partition, en particulier pour cacher une voix muette pendant un certain temps (figure 13).

Son usage est généralisé à tous les éléments et toutes les voix de la partition et permet des *montages* plus complexes (figure 14).

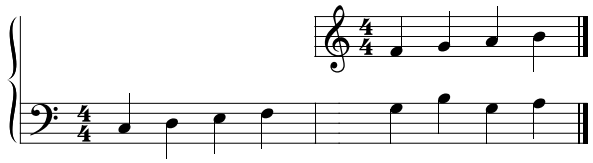


Figure 13. \staffOff classique

```
{
 [
  \clef \meter<"3/4"> a b/2 e/4
  \staffOff f g
 ],
 [
  \meter<"3/4"> \staffOff \clef c
  \staffOn d g | \staffOff f g \staffOn b
 ]
}
```

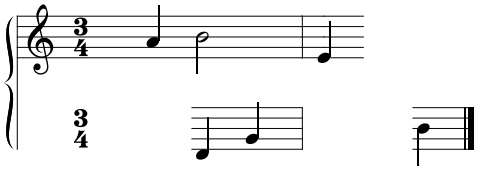


Figure 14. \staffOff plus complexe

L'implémentation graphique est le pendant de la description textuelle : tous les éléments inscrits après un tag \staffOff dans la description textuelle ne sont pas dessinés jusqu'au prochain \staffOn dans la séquence courante. Pour plusieurs éléments simultanés, c'est donc bien l'ordre dans la description textuelle qui indiquera le comportement à adopter. Ainsi :

```
[ \meter<"4/4"> \clef \staffOff
  a b \staffOn c ]
```

n'est pas équivalent à :

```
[ \staffOff \meter<"4/4">
  \clef a b \staffOn c ]
```

Quant à la portée, elle commence, ou s'arrête, aux positions correspondant au début des éléments suivant le tag.

3.8. Graphiques arbitraires

Le tag \symbol permet d'insérer des graphiques arbitraires dans la partition, sous forme de fichier image externe. Les formats graphiques supportés sont de type *png*, *jpg* ou *bmp*. Les paramètres de contrôle permettent d'ajuster la taille et la position de l'image dans la partition.

Sa syntaxe est la suivante :

```
1) \symbol<params>
2) \symbol<params>(notes)
  params :
    - filePath : chemin du fichier
    - position = [top | mid | bot] :
      position de l'image
    - size : taille de l'image
    - w/h : largeur/hauteur de l'image
```

Le tag \symbol peut être utilisé comme un tag simple ou comme un range-tag :

1. le symbole ne possède pas de durée mais prend de la place sur la portée, selon sa taille ;
2. le symbole s'applique aux notes indiquées (sans étirement).

Le fichier est indiqué par un chemin absolu ou relatif. Dans le second cas, le chemin est relatif au répertoire contenant le fichier source ou au répertoire *home* de l'utilisateur.

EXEMPLE

Sur la première portée, le symbole ne possède pas de durée, contrairement à la deuxième portée.

```
{
  [
    \meter<"4/4"> c f
    \symbol<file="silence.png", dx=-5,
        dy=-10>
    c d f
  ],
  [
    \meter<"4/4"> a d
    \symbol<file="ronds.png",
        position="bot"> (f g) g
  ]
}
```

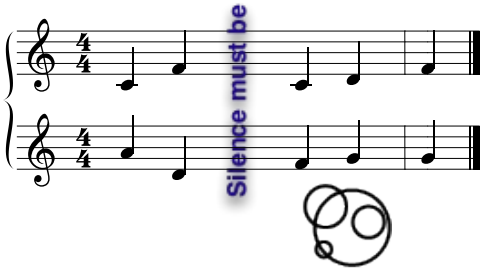


Figure 15. Graphiques arbitraires

4. CONTRÔLE DE RENDU

Après avoir exposé les nouvelles notations implémentées dans GUIDO, nous allons à présent nous intéresser aux améliorations et subtilités rajoutées aux notations existantes.

4.1. Nouveaux paramètres

De nouveaux paramètres ont été introduits pour plusieurs tags déjà existants, permettant un contrôle étendu du rendu graphique :

- `\staffFormat` : nouvelle possibilité de contrôle de l'épaisseur des lignes de la portée
- `\tuplet` : nouvelle possibilité de contrôle de l'épaisseur des lignes, de la taille du texte, etc.
- `\cresc` et `\decresc` : maintenant paramétrable en position, épaisseur, couleur.

On se reportera à la documentation utilisateur pour le détail de ces paramètres.

EXEMPLE

L'exemple suivant illustre ces améliorations et génère la partition de la figure 16.

```
{
  [ \tuplet<"-3-", lineThickness=0.4,
       bold="true", textSize=2, dy1=10> (
      \cresc<dm="fff", dx1=-5, dx2=5,
          dy=1, deltaY=5, color="red",
          size=1.7> (a a a) )
  ],
  [ \staffFormat<lineThickness=0.5>
    \decresc<thickness=0.8> (g d e f)
  ]
}
```

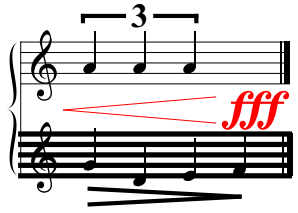


Figure 16. Contrôle de rendu étendu

4.2. Améliorations des *trilles*

```
\trill<params>(chord-series)
 params:
  - tr = [true | false]
  - anchor = [note | tr]
```

Un *trille* est un ornement musical qui consiste à alterner rapidement la note de base, sur laquelle est noté le *trille*, et la note située juste au-dessus.

Dans la version précédente de GUIDO, le *trille* à proprement parler n'était indiqué que par le symbole ***tr*** placé au-dessus de la note. Il paraissait pourtant important que la notation comprenne également la ligne ondulée du *trille* qui suit celui-ci et en indique la durée. Comme ce symbole était déjà en partie implémenté, il fallait pouvoir lui offrir plus de souplesse sans pour autant en complexifier la syntaxe.

De plus, nous avons voulu donner la possibilité de choisir de dessiner ou non le signe ***tr***, ainsi que de pouvoir changer l'ancrage de la ligne pour le déplacer sur la tête de la note. Cela fut implémenté grâce au *param* `tr` qui accepte comme valeurs `true` ou `false` et à `anchor` qui accepte comme valeurs `note` ou `tr` (figure 17).

Le reste des modifications graphiques pouvant être effectuées manuellement par l'utilisateur à travers les paramètres classiques de déplacement (`dx`, `dy`), de couleur (`color`), et de taille (`size`).

EXEMPLE

```
[ \trill<tr="false", anchor="note">
  ( {g} {a/2} ) ]
```

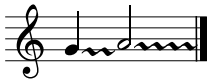


Figure 17. Cas du *trille* ancré à la tête de note, et ***tr*** optionnel

Enfin, une subtilité au niveau du contrôle du rendu a également été ajoutée. Il fallait définir le comportement quant aux notes liées. La solution adoptée est de dessiner la ligne du *trille* jusqu'à la fin de la dernière note liée si celle-ci provient d'une note plus longue découpée automatiquement, mais de s'arrêter normalement à la prochaine note si la liaison a été explicitée par l'utilisateur, qui pourra alors décider de répéter le *trille* sur cette autre note, ou non. (figure 18)

EXEMPLE

```
{
  [
    \meter<"2/4"> \trill({a} {a/2})
  ],
  [
    \meter<"2/4"> \trill(
      {a} \tie({a} {a})
    )
  ]
}
```

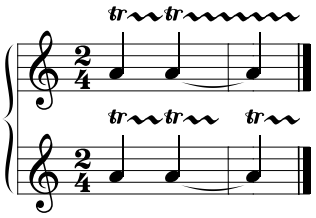


Figure 18. Cas du *trille* appliqué à des notes liées

4.3. *Glissandi* et accords

Un *glissando* entre deux accords soulève la question de la répartition des *glissandi* entre les notes des accords. La solution proposée est celle qui nous a paru la plus intuitive : c'est l'ordre des notes dans la description textuelle de l'accord qui détermine les relations entre les notes. Par exemple, pour 2 accords A et B, la première note de l'accord A sera liée à la première note de l'accord B, la seconde de A à la seconde de B, etc. (figure 19).

EXEMPLE

```
[ \glissando({e,a} {f,b} {a,d}) ]
```

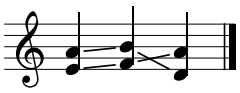
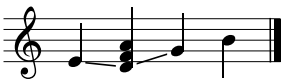


Figure 19. *glissandi* entre accords

L'unique problème avec ce choix peut se poser lors d'un conflit dans l'ordre imposé par le glissando précédent et par le suivant, comme dans le premier cas de l'exemple de la figure 20. Il est alors possible de faire appel à une seconde voix, et au tag `\staff<numéro de portée>` qui pourra créer sur une même portée d'autres *glissandi* en parallèle et indépendants des premiers (figure 20).

EXEMPLE

```
[ \glissando(e {d,f,a} g) b ]
```



```
{
  [ \glissando(e {d,f}) empty b ],
  [ \staff<1> empty \glissando(a g) empty ]
}
```

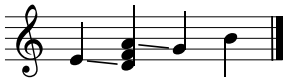


Figure 20. Cas de recours à une seconde voix pour le *glissando* (`empty` représente une note invisible.)

Enfin, lors de combinaisons de *glissandi* avec des accords ou des *clusters*, il nous a paru intéressant, comme on peut le voir dans [4] (p. 143) ou dans [16] (p. 61), de donner la possibilité de remplir l'espace entre ces *glissandi* grâce à un *param* `fill`. Une certaine flexibilité peut être trouvée dans le dessin grâce aux *params* graphiques `dx1`, `dx2`, `dy1`, `dy2`, ainsi que `thickness` qui permettent de modifier l'aspect du *glissando* (figure 21).

EXEMPLE

```
[
  \glissando<fill="true", dx1=-2, dx2=2,
      thickness=2.2> (
    \cluster({e,g} {c,b})
  )
  \glissando<fill="true">(
    {c,e,g} {a,c2,f1}
  )
]
```

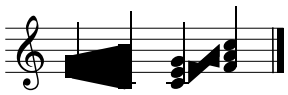


Figure 21. *Glissandi* entre accords et *clusters* avec *param* de remplissage

4.4. Combinaisons de *beams*

Pour finir, concernant le *feathered beaming* comme le *beaming* classique, il est maintenant possible de créer des hiérarchies de *beams* : c'est à dire d'englober plusieurs *beams* dans un plus grand, afin de les joindre par une barre principale commune (figure 22).

EXEMPLE

```
[
  \beam(
    \fBeam(c/8 e d f g/32)
    \fBeam(a/16 f e d c/64)
  )
]
```

Figure 22. *Feathered beams* englobés dans un plus grand

5. CONCLUSION

Le projet GUIDO se différencie des approches de type gravure musicale telles que Lilypond ou MuseScore (pour ne citer que les outils *libres*) en ce sens qu'il privilégie un rendu efficace et qu'il met en œuvre un grand nombre d'automatismes pour le calcul de la partition. Il est cependant plus limité en matière de complexité et de notations supportées.

Les extensions qui ont été présentées comblent pour partie le fossé entre le moteur GUIDO et les approches citées précédemment. Elles ont été réalisées principalement à la demande de compositeurs et au service de créations en cours. Elles s'intègrent dans le moteur existant dans le respect des différentes combinaisons du langage. Leur implémentation a soulevé des problèmes de design qui ont

été résolus en laissant à l'utilisateur le choix entre les différentes possibilités. Elles sont disponibles avec la version 1.52 de la librairie GUIDO. Le projet GUIDO est un projet *open source* hébergé sur Sourceforge [1] .

6. REFERENCES

[1] C. Daudin, Dominique Fober, Stephane Letz, and Yann Orlarey. La librairie guido – une boite à outils pour le rendu de partitions musicales. In ACROE, editor, *Actes des Journées d'Informatique Musicale JIM'09 – Grenoble*, pages 59–64, 2009.

[2] Dominique Fober, Yann Orlarey, and Stephane Letz. Inscore – an environment for the design of live music scores. In *Proceedings of the Linux Audio Conference – LAC 2012*, pages 47–54, 2012.

[3] M. Good. MusicXML for Notation and Analysis. In W. B. Hewlett and E. Selfridge-Field, editors, *The Virtual Score*, pages 113–124. MIT Press, 2001.

[4] E. Gould. *Behind Bars : The Definitive Guide to Music Notation*. Faber Edition Series. Faber Music Limited, 2011.

[5] Walter B. Hewlett. MuseData : Multipurpose Representation. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 402–447. MIT Press, 1997.

[6] Richard Hoadley. Calder's violin : Real-time notation and performance through musically expressive algorithms. In ICMA, editor, *Proceedings of International Computer Music Conference*, pages 188–193, 2012.

[7] H. Hoos, K. Hamel, K. Renz, and J. Kilian. The GUIDO Music Notation Format - a Novel Approach for Adequately Representing Score-level Music. In *Proceedings of the International Computer Music Conference*, pages 451–454. ICMA, 1998.

[8] H. H. Hoos and K. A. Hamel. The GUIDO Music Notation Format Specification - version 1.0, part 1 : Basic GUIDO. Technical report TI 20/97, Technische Universitat Darmstadt, 1997.

[9] David Huron. Humdrum and Kern : Selective Feature Encoding. In Selfridge-Field E., editor, *Beyond MIDI, The handbook of Musical Codes.*, pages 376–401. MIT Press, 1997.

[10] Smith Leland. SCORE. In *Beyond MIDI, The handbook of Musical Codes.*, pages 252–280. MIT Press, 1997.

[11] Han-Wen Nienhuys. Lilypond, automated music formatting and the art of shipping. In *Forum Internacional Software Livre 2006 (FISL7.0)*, 2006.

[12] Han-Wen Nienhuys and Jan Nieuwenhuizen. LilyPond, a system for automated music engraving. In *Proceedings of the XIV Colloquium on Musical Informatics (XIV CIM 2003)*, May 2003.

1. `http://guidolib.sf.net`

[13] G. Read. *Music notation : a manual of modern practice*. Crescendo book. Allyn and Bacon, 1969.

[14] Kai Renz. *Algorithms and Data Structures for a Music Notation System based on GUIDO Music Notation*. PhD thesis, Technischen Universität Darmstadt, 2002.

[15] E. Selfridge-Field. DARMS, Its Dialiects, and Its Uses. In *Beyond MIDI, The handbook of Musical Codes.*, pages 163–174. MIT Press, 1997.

[16] K. Stone. *Music Notation in the Twentieth Century : A Practical Guidebook*. W W Norton & Company Incorporated, 1980.