tikzanim
Créer une animation avec tikz

Frédéric Bonnaud

15 juin 2020

Table des matières

1 Introduction

tikzanim permet de réaliser simplement des animations en LaTeXà l'aide de tikz et animate en décrivant les différentes étapes de l'animation par un code tikz standard.
Une animation est constituée d'un certaine nombre d'étapes. Chaque étape dessine des transparents qui sont combinés les uns aux autres pour créer les images du morceau d'animation qu'elle constitue.

2 Les macros offertes par tikzanim

2.1 \tikzanim

Cette macro installe tout ce qu'il faut pour mettre en place l'animation : un environnement animateinline (pour créer l'animation), la macro \multiframe (pour créer les différents images) et un environnement \tikzpicture (pour dessiner les différentes images).
Syntaxe :

```
\tikzanim [options animateinline] {rafraîchissement} [options tikz] {étapes}
```

— [options animateinline] : les options qui seront passées à l'environnement animateinline.
Valeur par défaut : poster=last,controls

— {rafraîchissement} : le nombre d'images par seconde au début de l'animation. Il peut être modifié à chaque étape.

— [options tikz] : les options qui seront passée à l'environnement tikzpicture.
Valeur par défaut : pas d'options

— {étapes} : la suite des différentes étapes de l'animation.

2.2 \step

Cette macro définit une nouvelle étape de l'animation. `animate` définit une pile de transparents. Chaque image de l'animation est générée par l'affichage de toutes la pile de transparents à chaque instant. Le rôle de `\step` est de gérer la création (avec `tikz`) et la gestion cette pile (en générant un fichier timeline) pour créer les différentes images qui permettrons créer l'animation.
Au début de l'animation, la pile est vide. Un appel à `\step` va :

(1) Générer la prermière image de cette étape et la déposer sur la pile pour une durée de un image.
(2) Enlever la dernière image de la pile.
(3) Générer l'image suivante et la déposer sur la pile pour une durée de un image.
(4) Si ce n'est pas la fin de l'étape, on boucle vers (2).
(5) Si c'est la fin de l'étape, on laisse la dernière image pour une durée définie.

Syntaxe :
`\step[*] [rafraîchissement] {images} [durée] {initialisation} {dessin}`

— `*` : la version étoilée, vide la pile de transparents au début de l'étape.

— `[rafraîchissement]` : définit le nouveau nombre d'images par seconde.
Valeur par défaut : 0, c'est à dire le dernier nombre d'images par seconde définit par `\tikzanim` ou `\step`.

— `{images}` : définit le nombre de images que doit générer `\step`. Si ce nombre est 0. L'image est créée, mais aucune image n'est créée pour elle. L'image peut tout de même avoir une durée. Cela permet de dessiner dans des étapes différentes des objets qui auront des durées d'affichage différentes.

— `[durée]` : définit le temps pendant lequel la derniére image doit durer (en nombre d'images).
Valeur par défaut : 0, c'est à dire jusqu'à la fin de l'animation.

— `{initialisation}` : définit le code `tikz` d'initialisation. Comme l'animation est générée par étape, pour pouvoir réutiliser des éléments des étapes précédentes aux étapes suivantes, il faut que ces éléments soit définis. C'est dans le code d'initialisation qu'il faut le faire.
Pour que le code d'initialisation dépende du transparent en train d'être générer, on peut utiliser :

— `\framepos` qui vaut 0 au début de l'étape et varie de façon linéaire jusqu'à 1 à la fin de l'étape. Cette macro est gérée par `\step`.

— `\iframe` qui est égale au numéro du transparent en cours de génération. Cette macro est gérée par `\multiframe` du package `animate`.

Remarque : pour fixer une taille identique pour toutes les images, le code d'initialisation de la première étape devrait toujours contenir un appel à ***`\useasboundingbox`***

— `{dessin}` : définit le code `tikz` chargé de dessiner les différents transparents. Ce code peut utiliser tout ce qui se trouve dans les codes d'initialisation des étapes précédentes, ainsi que `\framepos` et `\iframe`.

Le code suivant :

```
% \usepackage{tikzanim}
\tikzanim{10}{
  % cette étape se prolongera sur 15 images
  % c'est à dire la moitié de l'étape suivante
  \step{30}[15]{
    % il faut initialiser A et B ici, car
    % la 'boundingbox' doit être utilisée
    % à l'étape suivante
    \coordinate(A)at(0,0) ;
    \coordinate(B)at(5,5) ;
    \useasboundingbox(A)rectangle(B) ;
  }{
    \draw(A)--($(A)!\framepos!(B)$) ;
  }
  % la seconde étape sera 2 fois plus rapide
  \step[20]{30}{
    % pas d'initialisation, C et D ne
    % sont pas utilisés par la suite
  }{
    \coordinate(C)at(0,5) ;
    \coordinate(D)at(5,0) ;
    \draw(C)--($(C)!\framepos!(D)$) ;
```

```
23   }
24 }
```

Crée l'animation :

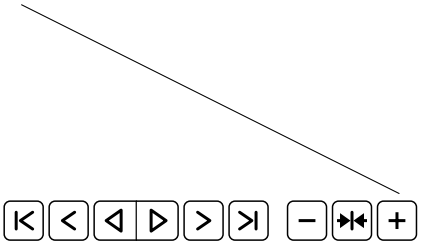

Schéma de fonctionnement :

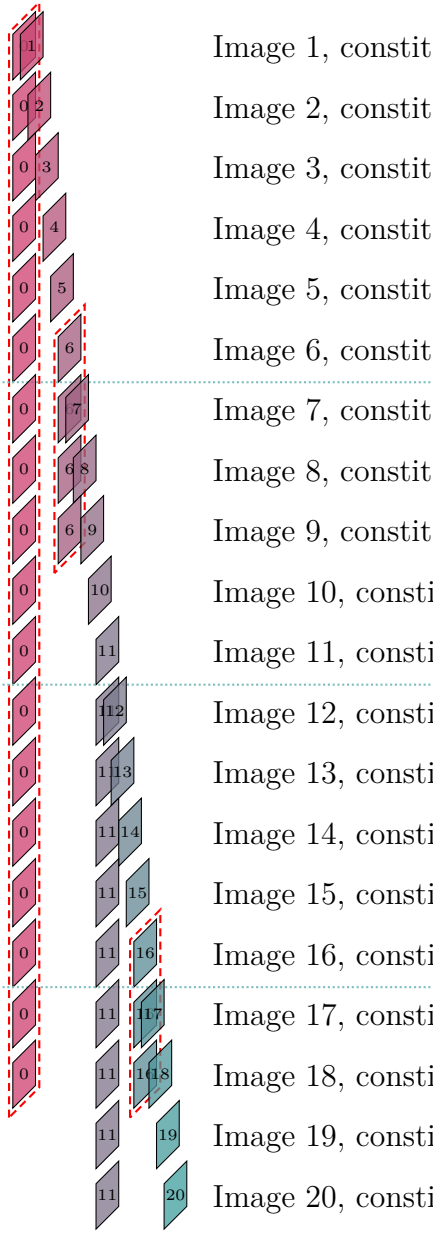

Image 1, constituée des transparents : 0,1
Image 2, constituée des transparents : 0,2
Image 3, constituée des transparents : 0,3
Image 4, constituée des transparents : 0,4
Image 5, constituée des transparents : 0,5
Image 6, constituée des transparents : 0,6

`\step{0}[18]{...}` ajoute 1 transparent qui dure 18 images, sans ajouter d'image
`\step{6}[4]{...}` ajoute 6 transparents et 6 images, le dernier dure 4 images

Image 7, constituée des transparents : 0,6,7
Image 8, constituée des transparents : 0,6,8
Image 9, constituée des transparents : 0,6,9
Image 10, constituée des transparents : 0,10
Image 11, constituée des transparents : 0,11

`\step{5}{...}` ajoute 5 transparents et 5 images

Image 12, constituée des transparents : 0,11,12
Image 13, constituée des transparents : 0,11,13
Image 14, constituée des transparents : 0,11,14
Image 15, constituée des transparents : 0,11,15
Image 16, constituée des transparents : 0,11,16

`\step{5}[3]{...}` ajoute 5 transparents et 5 images, le dernier dure 3 images

Image 17, constituée des transparents : 0,11,16,17
Image 18, constituée des transparents : 0,11,16,18
Image 19, constituée des transparents : 11,19
Image 20, constituée des transparents : 11,20

`\step{4}{...}` ajoute 4 transparents et 4 images

2.3 `\allframes`

Cette macro permet de dessiner sur tous les transparents.
Syntaxe :

```
\allframes {dessin}
```

— `{dessin}` : définit le code `tikz` chargé de dessiner sur tous les transparents. Ce code peut utiliser tout ce qui se trouve dans les codes d'initialisation des étapes précédentes, ainsi que `\framepos` et `\iframe`.

3 timeline

animate utilise un fichier timeline pour gérer l'affichage des transparents. tikzanim crée un fichier timeline par animation. Ils sont nommés : \jobname.tzc#.tln. Ils peuvent être utiliser à des fins de débuggage de l'animation.

4 Un exemple complet

```
\gdef\aLen{5}
\gdef\bLen{2}
\pgfmathsetmacro{\cLen}{sqrt(\aLen^2+\bLen^2)}
\tikzset{bleu/.style={fill=blue!30,fill opacity=0.5},rouge/.style={fill=red!30,fill opacity=0.5}}
```

Quelques initialisations.

```

\tikzanim[poster=0,controls]{1}{
```

On crée une animation à une image par seconde (au départ).
Puis, on crée les points servants à la construction, en même temps qu'un permier transparent qui ne sera affiché qu'à la prochaine image.

```
  \step{0}[1]{
    % initialisation
    \useasboundingbox(0,-\bLen)rectangle(\aLen+\bLen,\aLen) ;

    \coordinate(A)at(0,0) ;
    \coordinate(B)at(\aLen,0) ;
    \coordinate(C)at(\aLen,\aLen) ;
    \coordinate(D)at(0,\aLen) ;

    \coordinate(E)at(\aLen+\bLen,0) ;
    \coordinate(F)at(\aLen+\bLen,\bLen) ;
    \coordinate(G)at(\aLen,\bLen) ;

    \path[name path=EC](E)--(C) ;
    \path[name path=FG](F)--(G) ;
    \path[name intersections={of=EC and FG,by=H}] ;

    \coordinate(I)at(0,\aLen-\bLen) ;
    \coordinate(J)at($(A)+(H)-(G)$);
  }{
    % premier morceau
    \fill[bleu] (C) -- (D) -- (I) -- cycle ;
  }
```

On crée un nouveau transparent qui ne sera affiché qu'avec la prochaine image.

```
  \step{0}[12]{}{
    % deuxième morceau
    \fill[bleu] (I) -- (A) -- (J) -- cycle ;
  }
```

On crée encore un nouveau transparent qui ne sera affiché qu'avec la prochaine image.

```
  \step{0}[23]{}{
    % troisième morceau
    \fill[rouge] (E) -- (F) -- (H) -- cycle ;
  }
```

On crée un seul transparent, qui contient ce qui va rester affiché jusqu'à la fin (et qui donc ne bougera pas) et les 3 transparents précédants qui disparaîtront lorsque le mouvement de ce qu'ils contiennent commencera.

```
\step{1}{
}{
  % figure de départ sans les morceaux
  \draw (A) -- (B) -- (C) -- (D) --cycle ;
  \draw (B) -- (E) -- (F) -- (G) --cycle ;

  \fill[bleu] (I) -- (J) -- (B) -- (C) -- cycle ;
  \draw (J) -- (B) -- (C) ; \draw[densely dotted] (C) -- (I) -- (J) ;

  \fill[rouge] (B) -- (E) -- (H) -- (G) -- cycle ;
  \draw (H) -- (G) -- (B) -- (E) ; \draw[densely dotted] (E) -- (H) ;
}
```

Le mouvement commence, on fixe le nombre d'image par seconde à 5, et on génére 10 images, cette étape va donc prendre 2 secondes. Comme sa durée est de 1 image, cela signifie que le dernier transparent ne sera pas affiché sur les images qui suivent cette étape.

```
% déplacement du premier triangle
\step[5]{10}[1]{
  \coordinate(M)at($(C)!\framepos!(E)$);% Pt à #1 sur [CE]
  \coordinate(CM)at($(M)-(C)$) ;
  \coordinate(ICM)at($(I)+(CM)$);
  \coordinate(CCM)at($(C)+(CM)$);
  \coordinate(DCM)at($(D)+(CM)$);
}{
  % on affiche une flèche en plus du triangle pour bien voir d'où vient le triangle.
  \coordinate(S)at(barycentric cs:I=1,C=1,D=1);
  \coordinate(S')at(barycentric cs:ICM=1,CCM=1,DCM=1);
  \fill[bleu] (ICM) -- (CCM) -- (DCM) -- cycle ;
  \draw[densely dotted] (ICM) -- (CCM) ;
  \draw (CCM) -- (DCM) -- (ICM) ;
  \draw[-latex](S)--(S') ;
}
```

Comme dernier transparent ne dure qu'une image, le triangle final ne reste pas dessiné. On le dessine à nouveau.

```
% position finale
\step{1}{}{
  \fill[bleu] (ICM) -- (CCM) -- (DCM) -- cycle ;
  \draw[densely dotted] (ICM) -- (CCM) ;
  \draw (CCM) -- (DCM) -- (ICM) ;
}
```

On procède de même pour les 2 autres triangles.

```
% déplacement du deuxième triangle
\step{10}[1]{
  \coordinate(N)at($(I)!\framepos!(C)$);% Pt à #1 sur [IC]
  \coordinate(IN)at($(N)-(I)$) ;
  \coordinate(IIN)at($(I)+(IN)$);
  \coordinate(JIN)at($(J)+(IN)$);
  \coordinate(AIN)at($(A)+(IN)$);
}{
  \coordinate(T)at(barycentric cs:I=1,J=1,A=1);
  \coordinate(T')at(barycentric cs:IIN=1,JIN=1,AIN=1);
  \fill[bleu] (IIN) -- (JIN) -- (AIN) -- cycle ;
  \draw[densely dotted] (IIN) -- (JIN) ;
  \draw (JIN) -- (AIN) -- (IIN) ;
  \draw[-latex](T)--(T') ;
}
% position finale
\step{1}{}{
  \fill[bleu] (IIN) -- (JIN) -- (AIN) -- cycle ;
  \draw[densely dotted] (IIN) -- (JIN) ;
```

```
    \draw (JIN) -- (AIN) -- (IIN) ;
  }
  % déplacement du troisième triangle
  \step{10}[1]{
    \coordinate(P)at($(H)!\framepos!(J)$);% Pt à #1 sur [HJ]
    \coordinate(HP)at($(P)-(H)$) ;
    \coordinate(EHP)at($(E)+(HP)$);
    \coordinate(FHP)at($(F)+(HP)$);
    \coordinate(HHP)at($(H)+(HP)$);
  }{
    \coordinate(U)at(barycentric cs:E=1,F=1,H=1);
    \coordinate(U')at(barycentric cs:EHP=1,FHP=1,HHP=1);
    \fill[rouge] (EHP) -- (FHP) -- (HHP) -- cycle ;
    \draw[densely dotted] (EHP) -- (HHP) ;
    \draw (EHP) -- (FHP) -- (HHP) ;
    \draw[-latex](U)--(U') ;
  }
  % position finale
  \step{1}{}{
    \fill[rouge] (EHP) -- (FHP) -- (HHP) -- cycle ;
    \draw[densely dotted] (EHP) -- (HHP) ;
    \draw (EHP) -- (FHP) -- (HHP) ;
  }
}
```

L'animation :
Une démonstration visuelle du théorème de Pythagore.