Discrete & Continuous Models
& Applied Computational Science
ISSN 2658-7149 (online), 2658-4670 (print)

2023, 31 (2) 139–149
http://journals.rudn.ru/miph

Research article

UDC 004.92:004.928:519.67
DOI: 10.22363/2658-4670-2023-31-2-139-149
EDN: XKNIYV

Asymptote-based scientific animation

Migran N. Gevorkyan[1],
Anna V. Korolkova[1], **Dmitry S. Kulyabov**[1, 2]

[1] *RUDN University,*
6, Miklukho-Maklaya St., Moscow, 117198, Russian Federation
[2] *Joint Institute for Nuclear Research,*
6, Joliot-Curie St., Dubna, Moscow Region, 141980, Russian Federation

(received: April 5, 2023; revised: May 5, 2023; accepted: June 26, 2023)

Abstract. This article discusses a universal way to create animation using Asymptote the language for vector graphics. The Asymptote language itself has a built-in library for creating animations, but its practical use is complicated by an extremely brief description in the official documentation and unstable execution of existing examples. The purpose of this article is to eliminate this gap. The method we describe is based on creating a PDF-file with frames using Asymptote, with further converting it into a set of PNG images and merging them into a video using FFmpeg. All stages are described in detail, which allows the reader to use the described method without being familiar with the used utilities.

Key words and phrases: vector graphics, TeX, asymptote, scientific graphics

1. Introduction

In this paper we study the creation of animation animation using the vector graphics language Asymptote [1–4].

Asymptote is an interpreted language, that is a translator into the PostScript vector graphics language. Designed to create vector images for mathematical publications. It is closely integrated with the TEX system and is an integral part of the TEX Live [5] distribution. It has a `C`-like syntax, supports the creation of functions, custom data structures, and comes with an extensive set of modules for various tasks. Unlike PGF/TikZ [6], Asymptote is more imperative, so it is easier to implement complex program logic on it.

In the official documentation of this language, only a few paragraphs are devoted to the animation creation process and the user is referred to the source code examples located in the `animations` directory. Asymptote creates animation in two steps. At the first step, a multi-page PDF-file is created containing images that will become frames of future animation. Then, using the external utility ImageMagick [7] (command `convert`), this PDF-file is

converted into a GIF image. If the ImageMagick utility is not installed on the user's system, all examples will stop at creating a multi-page PDF-file with a set of images and a GIF image with animation will not be received.

In this article, we are considering a universal way to create animation in video format using the ffmpeg [8, 9] and Ghostscript [10] utilities. All external programs will be called explicitly from the command line. With the help of Asymptote, only a multi-page PDF-file with frames for the future video will be created.

The reader should be familiar with the basic capabilities of the Asymptote language. For an introduction to the basics of the language, we recommend the manual [11]. The information from it will be enough to understand this work. As an example, we chose the animation of the process of constructing epitrochoids and hypotrochoids. In the first part of the work, we will recall the definitions of these curves, some of their properties and reduce their construction to a composition of two rotations. In the second part of the article, we will describe in detail the implementation of their construction using Asymptote. And in the third part we will focus on the technical side of the issue and describe the process of creating a multi-page PDF-file, converting it into PNG images using Ghostscript and converting these images into video using ffmpeg.

2. Task description

Consider the task of animating the process of constructing cycloidal curves, namely hypotrochoids and epitrochoids. We will not use the parametric equation of these curves, but will reduce everything to the composition of two rotations applied to the starting point of the curve. This will better illustrate the capabilities of the Asymptote language.

2.1. Definition of epitrochoids

Epitrochoid is defined as a trajectory plotted by a fixed point P lying on a radial line of circle with radius r, which rolls along the *outer side* of the circle with radius R (figure 1). The parametric equation of the curve has the following form:

$$\begin{cases} x(t) = (R+r)\cos(\varphi) - d\cos\left(\dfrac{R+r}{r}\varphi\right), \\ y(t) = (R+r)\sin(\varphi) - d\sin\left(\dfrac{R+r}{r}\varphi\right), \end{cases}$$

where d is the distance from the center of the rolling circle to the point of the curve, φ is the angle of rotation of the rolling circle relative to the axis Ox.

Let us introduce the coefficient $k = r/R$, then it will possible to change the parameterization and the equation will take the form:

$$\begin{cases} x(t) = R(k+1)\cos(kt) - d\cos((k+1)t), \\ y(t) = R(k+1)\sin(kt) - d\sin((k+1)t), \end{cases}$$

where the parameters t and φ are related as $\varphi = kt$.

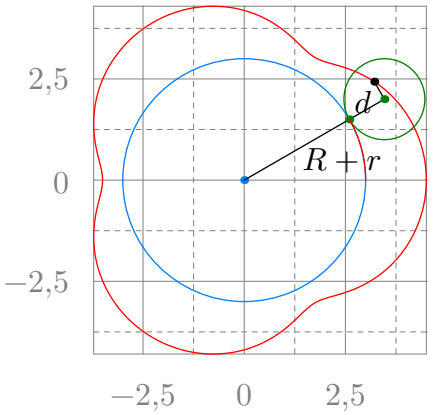

Figure 1. $R = 3, r = 1, d = 1/2$

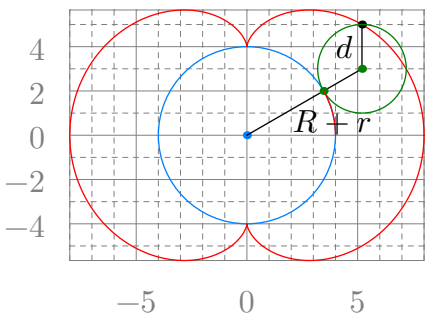

Figure 2. $R = 4, r = 2, d = 2$

Some special cases of epitrochoids have proper names. So for $r = R$, *Pascal's snail* is obtained, for $d = R + r$ — *rosy curve* or *rose*, and for $d = r$ — *epicycloid* (figure 2).

2.2. Definition of a hypotrochoid

Hypotrochoid is the trajectory described by a fixed point P on a radial straight circle of radius r, which rolls along the *inner* side of the circle of radius R (figure 3). The parametric equation of the curve has the following form:

$$\begin{cases} x(t) = (R - r)\cos(\varphi) + d\cos\left(\dfrac{R - r}{r}\varphi\right), \\ y(t) = (R - r)\sin(\varphi) - d\sin\left(\dfrac{R - r}{r}\varphi\right), \end{cases}$$

where, as in the case of the epitrochoid, d is the distance from the center of the rolling circle to the point P. In particular, for $d = r$, *hypocycloid* is obtained (figure 4).

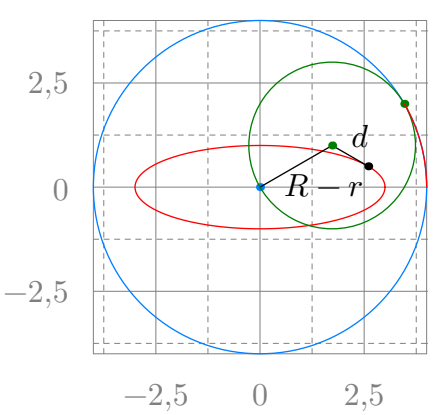

Figure 3. $R = 4, r = 2, d = 1$

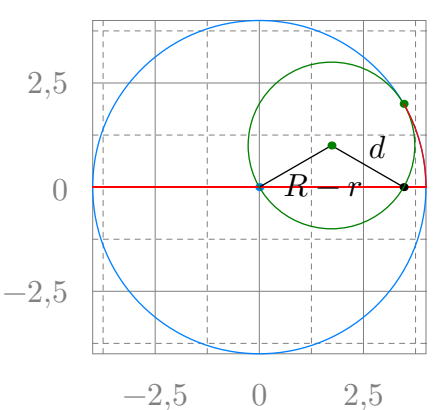

Figure 4. $R = 4, r = 2, d = 2$

It is also possible to parameterize $\varphi = kt$, where $k = r/R$, then the equation will take the form:

$$\begin{cases} x(t) = R(1-k)\cos(kt) + d\cos((1-k)t), \\ y(t) = R(1-k)\sin(kt) - d\sin((1-k)t). \end{cases}$$

2.3. Reducing the problem to a composition of turns

The construction of cycloidal curves begins by specifying two circles: a fixed circle of radius R centered at point O_R and a moving circle of radius r centered at point O_r.

A fixed circle will be conventionally called "large", and a moving one — "small", since usually $R > r$. On the radial line of a small circle, we fix the point of the curve P_0.

From the definition of hypotrochoids and epitrochoids, it follows that a motion $T(\varphi)$ is performed over the point P_0, consisting of a composition of two turns (figures 5–10):

1. $T_1(\varphi)$ — rotation around the point O_R by the angle φ, at which the point O_r turns into O'_r, and the point P_0 into the point $P_{1/2}$;
2. $T_2(\theta(\varphi))$ — rotation around the point O'_r by the angle θ, at which $P_{1/2}$ turns into P_1.

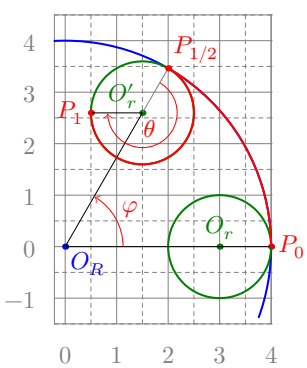

Figure 5. Hypocycloid $d = r$

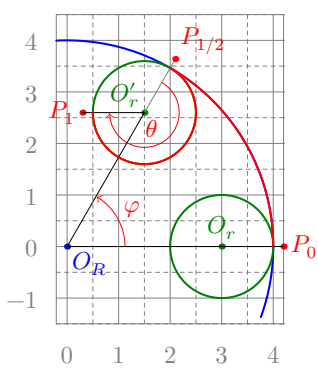

Figure 6. Hypotrochoid $d > r$

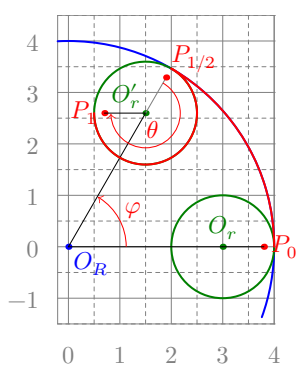

Figure 7. Hypotrochoid $d < r$

The rotation angle θ is related to the angle φ. A small circle must travel a distance equal to the length of the arc $PP_{1/2}$, which means that the lengths of the arcs $PP_{1/2}$ and $P_{1/2}P_1$ are equal.

$$|PP_{1/2}| = R\varphi = |P_{1/2}P_1| = \theta r \Rightarrow \theta = \frac{R\varphi}{r} = \frac{\varphi}{k}, \quad k = r/R.$$

Thus, to construct a curve, it is enough to set the parameters R, r and d, the initial positions of the circles and the points P_0. It is usually assumed that the center of O_R coincides with the origin, and the center of O_r lies on the Ox axis.

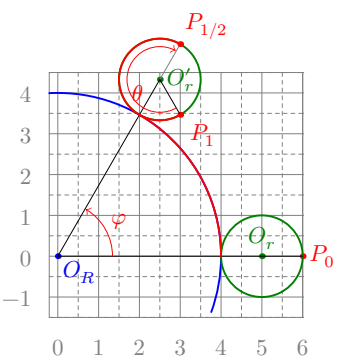

Figure 8. Epicycloid $d = r$

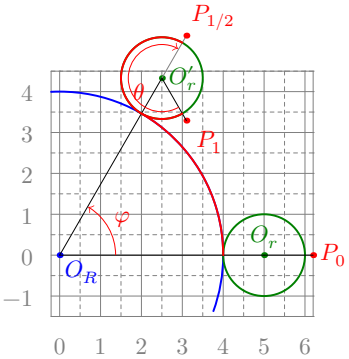

Figure 9. Epitrochoid $d > r$

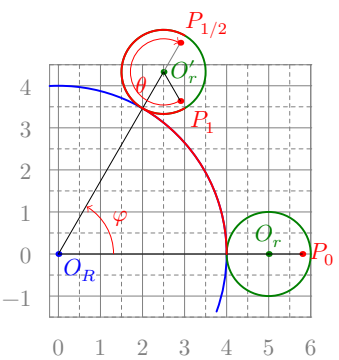

Figure 10. Epitrochoid $d < r$

Then the coordinates of the center O_r are calculated as:

$$\mathbf{OO}_r = \mathbf{OO}_R + (R + s \cdot r, 0)^T, \quad s = \begin{cases} +1, \text{ if epitrochoid,} \\ -1, \text{ if hypotrochoid.} \end{cases} \tag{1}$$

And the coordinates of the curve point are P_0: $\mathbf{OP}_0 = \mathbf{OO}_r + (d, 0)^T$.

Now, to find any point of the curve, it is enough to act on P_0 by converting $T(\varphi) = T_2(\varphi/k) \circ T_1(\varphi)$ setting the required value to φ. If it is necessary to construct a set of points, then taking a sufficiently small step φ, one can consistently act on the point P_0 by transformations $T(i\varphi), i = 1, 2, \ldots, n$:

$$P_0 \overset{T(\varphi)}{\longrightarrow} P_1,\ P_0 \overset{T(2\varphi)}{\longrightarrow} P_2,\ P_0 \overset{T(3\varphi)}{\longrightarrow} P_3,\ P_0 \overset{T(4\varphi)}{\longrightarrow} P_4,\ , \ldots, P_0 \overset{T(n\varphi)}{\longrightarrow} P_n,\ .$$

3. Implementation on Asymptote

Below we present the source code of the program in the Asymptote language and comment on its key points:

```
include "config.asy";

import animation;
import graph;

unitsize(1cm);
size(10cm, 10cm);

string ssign(int d) {
  return d > 0 ? "+" : "-";
}

transform T1(real phi, pair O_R) { // (2)
  return rotate(angle=phi, z=O_R);
}

transform T2(real phi, int sign, real k, pair O_r) { // (4)
```

```
  return rotate(angle=sign*phi/k, z=O_r);
}

pen BigCircle = blue;
pen littleCircle = deepgreen;
pen curve = red;

int sign = -1; // (6)
real R = 4.0;
real r = 2.0;
real d = 2.0;
real k = r/R;
int N = 100; // (8)
pair O_R = (0, 0);
int turns = 1; // (10)
usersetting(); // (12)

pair O_r = O_R + polar(R + sign * r, 0); // (14)
pair P = O_r + polar(d, 0);  // (16)
pair Q = O_r - sign * polar(r, 0);  // (18)

guide xcycloid;
transform T;

animation A; // (20)
A.global = true;

draw(circle(c=O_R, r=R), p=BigCircle); // (22)
dot(O_R, p=BigCircle);

for(real phi: uniform(0, 360turns, N)) {
  save(); // (24)
  T = T1(phi, O_R) * T2(phi, sign, k, O_r); // (26)
  xcycloid = xcycloid -- T*P; // (28)
  draw(xcycloid, p=1bp+curve); // (30)
  dot(T*P, L=Label("P", align=NW)); // (32)
  draw(O_R -- T*O_r, L=Label("R"+ssign(sign)+"r")); // (34)
  draw(T*O_r -- T*P, L=Label("d"));
  draw(circle(c=T*O_r, r=r), p=littleCircle); // (36)
  dot(T*O_r, p=littleCircle);
  dot(T1(phi, O_R)*Q, p=littleCircle); // (38)

  include "axes.asy"; // (40)

  A.add(); // (42)
  restore(); // (44)
}

A.movie(); // (46)
currentpicture.erase();
```

This program creates a multi-page PDF-file, each page of which is a future frame of the video. The main work on calculating the points of the curve is performed by the functions `T1` (2) and `T2` (4). These functions are defined for convenience, so that the code reflects the above formulas as much as possible. All the work is done by the built-in function `rotate`, which allows you to determine rotation around an arbitrary point (argument `z`) by an arbitrary angle value in degrees (argument `angle`).

Next, we set a set of variables-parameters (6). The variable `sign` is s from the formula (1), and the rest correspond to their mathematical notation. The variable `N` (8) sets the number of calculated points and, as a result, frames in the future video. The variable `turns` (10) sets the number of complete turns around the center of O_R. Calling the built-in function `usersetting` (12) to override the value of any variable specified above via the command line argument `-u`.

Then, based on the above-defined parameters, the coordinates of the initial position of the center of the moving circle O_r (14), the points of the curve P (16) and the point of tangency Q of the moving circle with the stationary (18) are calculated.

Next, an object `A` (20) is created, into which animation frames will be recorded (objects of the type `picture` or `frame`). `A` has several fields, in particular the `global` field of the `bool` type allows you to enable and disable saving the created images as an array in RAM and writing them as files to disk only after they are all built.

The curve points are calculated in a loop, but before that, a fixed circle (22) and its center are drawn. Then, at the beginning of each iteration of the cycle, all the current stationary elements of the image are saved (object `picture`) (24), all movable elements are built, the resulting image is added to the structure `A` (42) and the image state is reset (44) to the one that was at the time of (24). The process continues until all frames are drawn and saved to `A`.

As the cycle progresses, the angle φ changes from 0 to $2\pi n$ (in degrees). At each step, the rotation transformation $T(\varphi)$ is calculated (28), applied to the starting point of P and added to the path (`guide`) `xcycloid` (28). With each iteration of the loop, new points are added to the path `xcycloid` and the curve grows.

The following drawing commands follow:

— of the already calculated part of the curve (30);
— of the new point position P (32);
— of a segment of length $R + s \cdot r$ (34) connecting the center of O_R to the new position of the center of O_r, as well as a segment of length d connecting the new center of O_r to the point P of the curve;
— directly the moving circle itself in its new position (36) and its center;
— touch point Q (38);
— coordinate grid, the settings of which are placed in a separate file (40).

Finally, after working out the loop, all created frames are recorded in a PDF-file. To do this, Asymptote sequentially creates separate PDF-files for each frame, then adds text processed by LaTeX (in our case LuaLaTeX) to them.

It is this procedure that takes the main time of the program, the calculations themselves practically do not take up time in comparison with this.

We also note the peculiarity of the Asymptote syntax, which allows omitting the `*` operator when multiplying numeric literal constants and variables, for example `360turn` (24).

4. Creating a video clip

4.1. Launching Asymptote

To run the program discussed above, run the following command

```
asy -noV -nobatchView -f pdf -globalwrite -u
  ↪ 'R=3;r=1;d=1;N=100' xcycloid.asy -o video/xcycloid.pdf
```

The source code file `xcycloid.asy` is started for execution and as a result the file `xcycloid.pdf` will be created. Consider the options used:

— Options `-noV` and `-nobatchView` prevent the newly created image from opening automatically. The `-noV` option disables this function when executed from the command line, and `-nobatchView` when executing the script (as in our case).

— Option `-f pdf` indicates that you should immediately create a PDF-file, bypassing the postscript-file stage.

— Option `-globalwrite` makes it possible to save the file `xcycloid.pdf` to any directory (in our case `video`), and not only to the one where the source file `xcycloid.asy` is located.

— Option `-u` allows you to interact with the `usersetting()` function and pass the values of variables inside the program. So we pass the values `R=3`, `r=1`, `d=1` and `N=100`. This feature allows you to use a single source code file to build multiple images, flexibly adjusting any parameters. Note that this parameter takes exactly a text string, which is then processed by the `usersetting()` function, so the passed parameters must be taken in quotation marks.

4.2. Converting to PNG using GhostScript

To convert the resulting multipage file into a video format, it is necessary to convert its pages into bitmaps. To do this, we suggest using the GhostScript [10] program. It is available for both Windows and Unix systems (GNU/Linux, macOS). It also comes with the TEXLive [5] distribution, as does Asymptote.

To convert a PDF-file, run the command

```
gs -sDEVICE=png16m -r600 -o video/xcycloid-%04d.png
  ↪ video/xcycloid.pdf
```

In the case of using GhostScript from the TEXLive distribution, you should call `gs` using the `rungs` script, which is located:

— in the directory `texlive\2023\bin\win32` in the case of Windows OS,

— in the `texlive/2023/bin/x86_64-linux` in the case of GNU/Linux.

The `2023` directory corresponds to the version of the TEXLive distribution and may differ.

4.3. Creating a video using FFmpeg

The process of gluing the resulting bitmap images into one video clip is carried out using FFmpeg [9]. This program is a command-line utility and has extensive functionality and, as a result, a huge number of options and settings. Let's give an example of creating a video clip from the PNG images generated in the previous step and give an explanation of the parameters used:

```
ffmpeg -r 30 -f image2 -start_number 1 -i
↪ video/xcycloid-%04d.png -c:v libx264 -vf
↪ "pad=ceil(iw/2)*2:ceil(ih/2)*2" video/xcycloid.mp4
```

— Parameter `-r` sets the frame rate.
— Parameter `-f` sets the format of the input file.
— Since a lot of files are submitted to the input, you should specify the format of their names. The same notation is used as in the case of `gs`. The `-start_number` parameter sets the starting number.
— Parameter `-c:v` allows you to specify the video encoder used. In our case `libx264`, but many other formats are supported.
— The important parameter `-vf` sets the filter that is applied to the processed frame. In our case, we round the width and height of the frame in pixels to an even number. After converting to PNG, the width and height of the image may be odd, which is unacceptable for the vast majority of encoders. The specified filter allows you to avoid this error and rescale the frame by ffmpeg.

At the output we will get a video packed in a container `mp4`. The x264 format we have chosen is widespread and can be played by any browser.

5. Conclusion

We have analyzed in detail the way to create vector graphics animation on a plane using the Asymptote language. This aspect of this language is poorly covered in the official manual and, in our opinion, this article fills this gap. Although the result is a video clip containing bitmaps, but thanks to the vector source (PDF), you can increase the resolution of the video almost limitlessly. It should also be noted that this method of creating animation is universal, since almost any data visualization tool can be used to create a set of image frames. FFmpeg does all the work on creating a video file.

Acknowledgments

This paper has been supported by the RUDN University Strategic Academic Leadership Program.

References

[1] O. Shardt and J. C. Bowman, "Surface parameterization of nonsimply connected planar Bézier regions," *Computer-Aided Design*, vol. 44, no. 5, 484.e1–484.e10, May 2012. DOI: 10.1016/j.cad.2011.05.010.

[2] J. C. Bowman, "Asymptote: Interactive TEX-aware 3D vector graphics," *TUGboat*, vol. 31, no. 2, pp. 203–205, 2010.

[3] J. C. Bowman and A. Hammerlindl, "Asymptote: A vector graphics language," *TUGboat*, vol. 29, no. 2, pp. 288–294, 2008.

[4] J. C. Bowman. "Asymptote: The Vector Graphics Language." (May 2023), [Online]. Available: `https://asymptote.sourceforge.io/`.

[5] "TeX Live." (2023), [Online]. Available: `https://www.tug.org/texlive/`.

[6] T. Tantau and H. Menke. "PGF/TikZ." (2023), [Online]. Available: `https://ctan.org/pkg/pgf`.

[7] "ImageMagick." (Jun. 12, 2020), [Online]. Available: `https://imagemagick.org`.

[8] S. Tomar, "Converting video formats with FFmpeg," *Linux Journal*, vol. 2006, no. 146, p. 10, 2006.

[9] "FFmpeg Website." (2023), [Online]. Available: `https://ffmpeg.org/`.

[10] "Ghostscript Website." (2023), [Online]. Available: `https://www.ghostscript.com/`.

[11] C. I. Staats. "An Asymptote tutorial." (2015), [Online]. Available: `https://math.uchicago.edu/~cstaats/Charles_Staats_III/Notes_and_papers_files/asymptote_tutorial.pdf`.

For citation:
M. N. Gevorkyan, A. V. Korolkova, D. S. Kulyabov, Asymptote-based scientific animation, Discrete and Continuous Models and Applied Computational Science 31 (2) (2023) 139–149. DOI: 10.22363/2658-4670-2023-31-2-139-149.

Information about the authors:
Migran N. Gevorkyan — Candidate of Sciences in Physics and Mathematics, Associate Professor of Department of Applied Probability and Informatics of Peoples' Friendship University of Russia named after Patrice Lumumba (RUDN University) (e-mail: `gevorkyan-mn@rudn.ru`, phone: +7 (495) 955-09-27, ORCID: https://orcid.org/0000-0002-4834-4895)

Anna V. Korolkova — Candidate of Sciences in Physics and Mathematics, Associate Professor of Department of Applied Probability and Informatics of Peoples' Friendship University of Russia named after Patrice Lumumba (RUDN University) (e-mail: `korolkova-av@rudn.ru`, phone: +7(495) 952-02-50, ORCID: https://orcid.org/0000-0001-7141-7610)

Dmitry S. Kulyabov — Doctor of Sciences in Physics and Mathematics, Professor of the Department of Applied Probability and Informatics of Peoples' Friendship University of Russia named after Patrice Lumumba (RUDN University); Senior Researcher of Laboratory of Information Technologies, Joint Institute for Nuclear Research (e-mail: `kulyabov-ds@rudn.ru`, phone: +7 (495) 952-02-50, ORCID: https://orcid.org/0000-0002-0877-7063)

УДК 004.92:004.928:519.67
DOI: 10.22363/2658-4670-2023-31-2-139-149
EDN: XKNIYV

Научная анимация на основе Asymptote

М. Н. Геворкян[1], **А. В. Королькова**[1], **Д. С. Кулябов**[1,2]

[1] *Российский университет дружбы народов, ул. Миклухо-Маклая, д. 6, Москва, 117198, Россия*
[2] *Объединённый институт ядерных исследований, ул. Жолио-Кюри, д. 6, Дубна, Московская область, 141980, Россия*

Аннотация. В статье рассматривается универсальный способ создания анимации с помощью языка для создания векторной графики Asymptote. В сам язык Asymptote встроена библиотека для создания анимации, однако практическое её использование осложнено крайне кратким описанием в официальной документации и нестабильной работой существующих примеров. Целью статьи является устранение данного пробела. Излагаемый нами способ основывается на создании PDF-файла с кадрами с помощью Asymptote с дальнейшей конвертацией его в набор PNG-изображений и склейкой их в видео с помощью FFmpeg. Все этапы подробно описываются, что даёт возможность читателю использовать изложенный метод, не будучи знакомым с используемыми утилитами.

Ключевые слова: векторная графика, TeX, asymptote, научная графика