Podpora poznámok PG3 14. 10. 2020, animácia, výpočty (Szirmay-Kalos)

The **main problems in computer animation** are as follows
(Salomon, The Computer Graphics Manual , 2011, p. 930n):

1.
How to display on the screen only those parts of the scene that would be seen by an actual camera located at a certain point. This involves general perspective projection and clipping. In computer animation, the term *camera* replaces the word *observer*. This term refers to what is displayed on the screen (what we want the camera to see). In the computer, the camera is represented by several numbers describing its position, direction of view, an "up" direction, the distance *k* between it and the projection screen, and the two viewing half-angles *h* and *v* (Section 6.10).

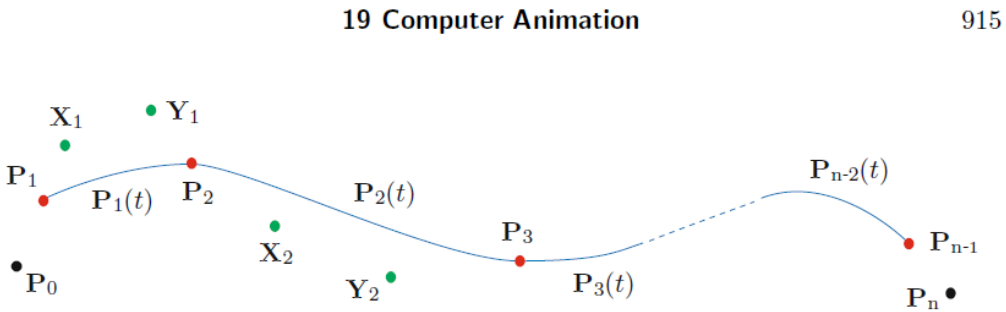


Figure 19.1: An Interpolating Bézier Curve for $n + 1$ Points.

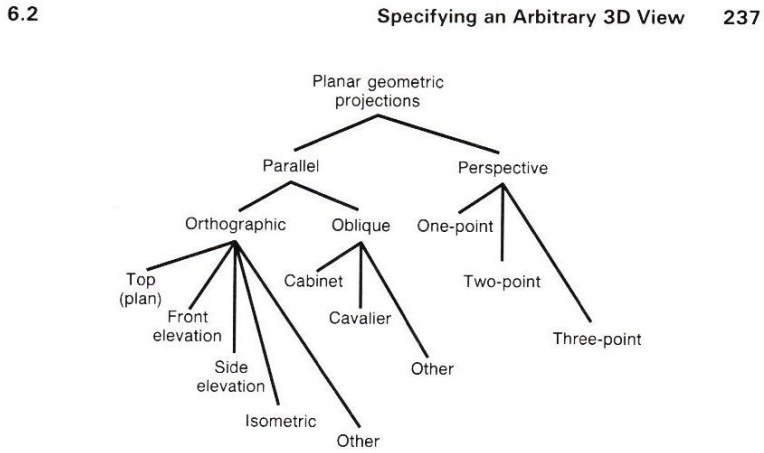


Fig. 6.13 The subclasses of planar geometric projections. *Plan view* is another term for a top view. *Front* and *side* are often used without the term *elevation*.

Možnosti výpočtu dĺžky sú napr. linearizácia lomenou čiarou, numerická integrácia, viaceré reprezentácie:
http://www.mathwords.com/a/arc_length_of_a_curve.htm

Animované opakovanie algoritmov viditeľnosti (priemyselný štandard je z-buffer) integrovala z diplomovky Zuzany Černekovej pod vedením Silvestra Czannera Kristína Vojtová v „našom" e-learningu
http://www.st.fmph.uniba.sk/~vojtova5/PG/index.html

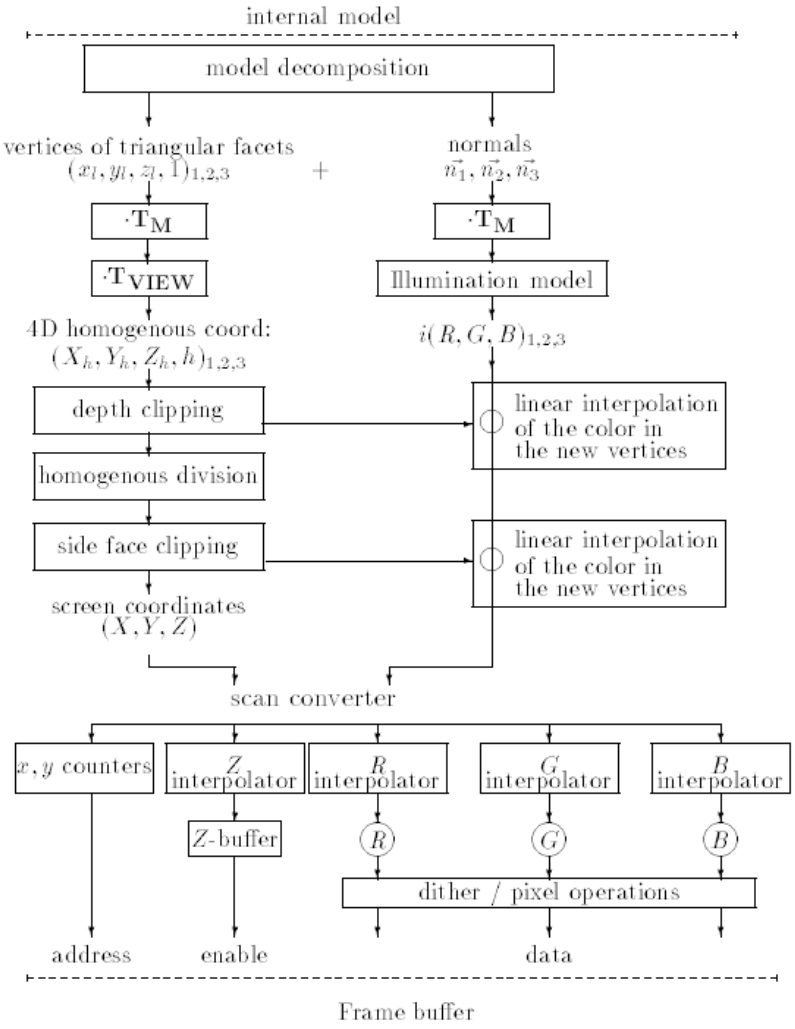


Figure 8.2: Data flow of shaded image synthesis

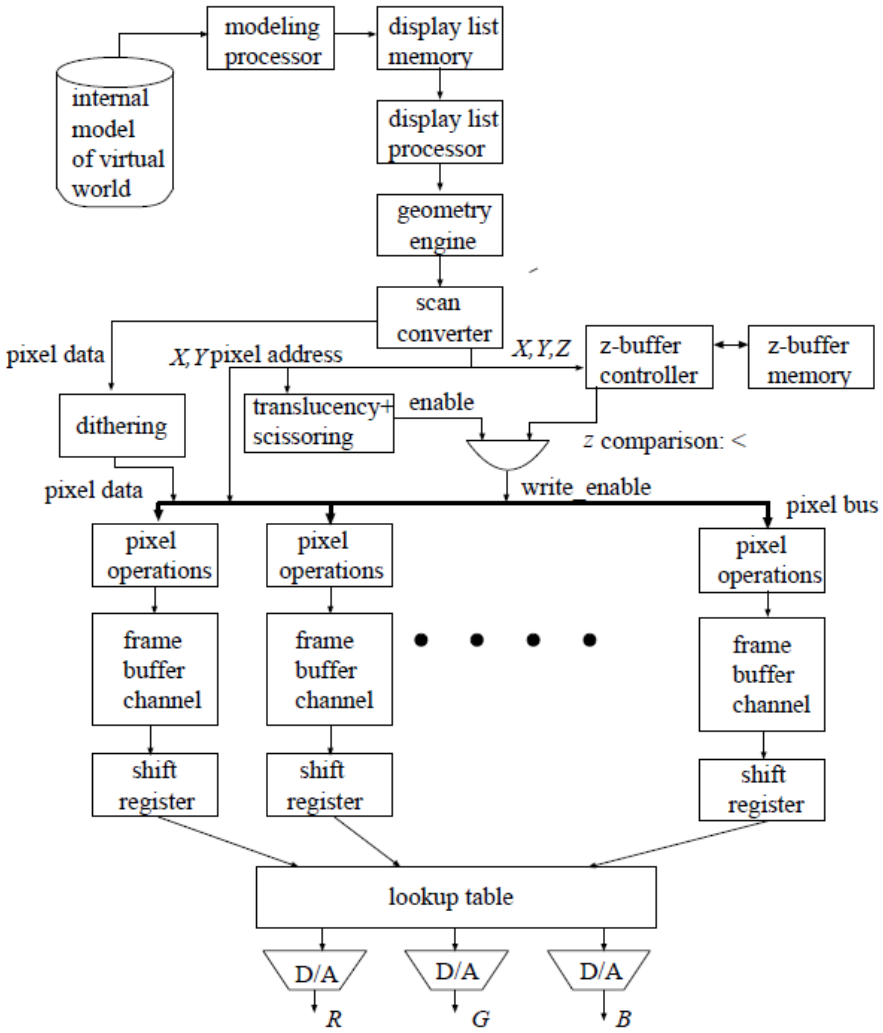


Figure 8.3: Architecture of z-buffer, Gouraud-shading graphics systems

Dátové toky pri výpočte tieňovania a architektúra grafického systému na riešenie viditeľnosti metódou z-buffer (Szirmay-Kalos).

2. How to move the camera along any desired path and rotate it during movement so it always points to the center of interest (generally a different direction in each frame). Its "up" direction may also have to be rotated to achieve the desired animation effects.

The images are shown one by one on the screen allowing a short time for the user to have a look at each one before the next is displayed.

Supposing that the objects are defined in their respective local coordinate system, the position and orientation of the particular object is given by its modeling transformation. Recall that this modeling transformation places the objects in the global world coordinate system determining the relative position and orientation from other objects and the camera.

The camera parameters, on the other hand, which include the position and orientation of the 3D window and the relative location of the camera, are given in the global coordinate system thus defining the viewing transformation which takes us from the world to the screen coordinate system.

Both transformations can be characterized by 4×4 homogeneous matrices. Let the time-varying modeling transformation of object o be $\mathbf{T}_{\mathbf{M},o}(t)$ and the viewing transformation be $\mathbf{T}_{\mathbf{V}}(t)$.

A simplistic algorithm of the generation of an animation sequence, assuming a built-in timer, is:


```
Initialize Timer( t_start );
do
    t = Read Timer;
    for each object o do Set modeling transformation: T_M,o = T_M,o(t);
    Set viewing transformation: T_V = T_V(t);
    Generate Image;
while t < t_end;
```

In order to provide the effect of continuous motion, a new static image should be generated at least every 60 msec. If the computer is capable of producing the sequence at such a speed, we call this **real-time animation**, since now the timer can provide real time values. With less powerful computers we are still able to generate continuous looking sequences by storing the computed image sequence on mass storage, such as a video recorder, and replaying them later. This technique, called **non-real-time animation**,

3. How to move the scene along another path (mathematically this is the same as moving the camera) and move parts of the scene in different ways (imagine a person walking, moving hands and feet in a complex pattern).

4. A typical *sequence* in a computer-generated piece of animation involves a camera moving smoothly along a curved path around a scene composed of objects. The objects may also move at the same time. Creating such a sequence requires the following tasks:

5. Defining the camera path. This may be a long, complex curve but the software should be able to follow it and to stop at many points (frames) for a snapshot. The frames should be equally spaced if uniform camera speed and smooth animation are important. Special effects may require the camera to accelerate or to slow down. The

problem is that a typical parametric curve **P**(*t*) has variable velocity; varying *t* in equal increments advances unequal segments on the curve.

6. At each point, the camera may have to be rotated so that it points in the right direction (normally directly at the scene, but sometimes off it). This is where *spherical interpolation* is used (Section 19.5).

7. When the camera is properly positioned, a snapshot is taken. This is done by projecting the scene (or part of it) on the screen, which is assumed to be perpendicular to the line of sight of the camera, at a distance of *k* units from it. The *y* axis of the screen should be in the "up" direction of the camera. Perspective projection is normally used, since an image generated by other types of projection may look unnatural.

8. The objects constituting the scene may also have to be moved and rotated (imagine a camera flying over a moving train). This task can use the techniques and tools developed for tasks 1 and 2. In fact, the case where only the camera moves and the objects of the scene are stationary is special and is referred to as a "walk-through" or a "flyby."

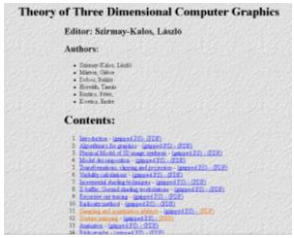
Modeling. **A mathematical model of an object is a collection of points and curves**. If the object is three-dimensional, the model also contains surface information. The first step in rendering an object is to display its mathematical model as a wireframe, (Salomon, p. 4.)

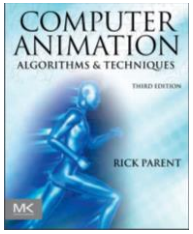
---

*Kuriózne sa animačné princípy dajú využiť aj vo vyučovaní, "lecturing is that mysterious process by means of which the contents of the note-book of the professor are transferred through the instrument of the (...) pen to the note-book of the student without passing through the mind of either", in **Adapting the twelve principles of classic animation to lectures**, (Gilardi et al., 2015)*

* * *

Autori a knihy, Szimay-Kalos, Rick Parent, John Lasetter + Luxo


laws of physics, specifically, **Newton's law** stating that the acceleration of masses is proportional to the resultant driving force. Let a point of object mass have positional vector $\vec{r}(t)$ at time t, and assume that the resultant driving force on this mass is $\vec{D}$. The position vector can be expressed by the modeling transformation and the position in the local coordinates ($\vec{r}_L$):

$$\vec{r}(t) = \vec{r}_L \cdot \mathbf{T_M}(t). \tag{13.1}$$

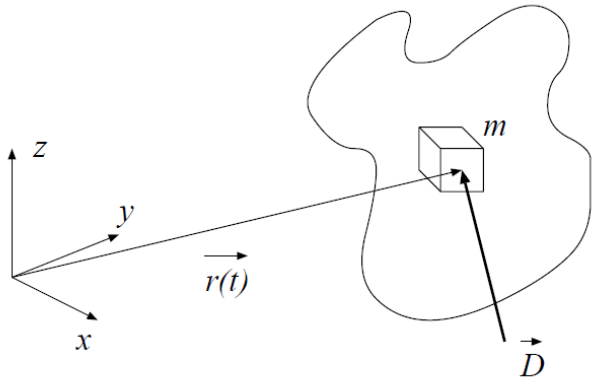


Figure 13.1: Dynamics of a single point of mass in an object

uous. To summarize, the illusion of realistic motion requires the elements of the transformation matrices to have finite and continuous second derivatives. Functions meeting these requirements belong to the C^2 family (C stands for parametric continuity, and superscript 2 denotes that the second derivatives are regarded). The C^2 property implies that the function is also of type C^1 and C^0.

The crucial problem of animation is the definition of the appropriate matrices $\mathbf{T_M}(t)$ and $\mathbf{T_V}(t)$ to reflect user intention and also to give the illusion of realistic motion. This task is called **motion control**.

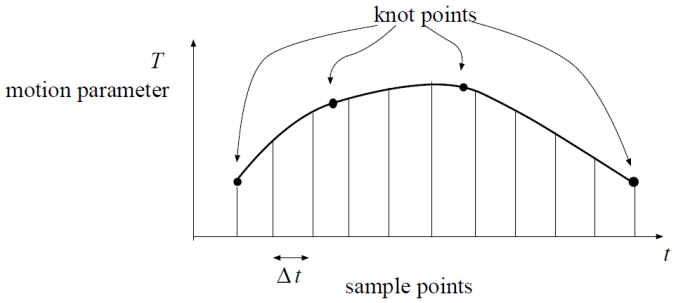


Figure 13.2: Motion control by interpolation

13.1 Interpolation of position-orientation matrices

As we know, arbitrary position and orientation can be defined by a matrix of the following form:

$$\begin{bmatrix} & & 0 \\ & \mathbf{A}_{3\times3} & 0 \\ & & 0 \\ \mathbf{q^T} & & 1 \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} & a_{13} & 0 \\ a_{21} & a_{22} & a_{23} & 0 \\ a_{31} & a_{32} & a_{33} & 0 \\ q_x & q_y & q_z & 1 \end{bmatrix}. \tag{13.3}$$

Vector $\mathbf{q^T}$ sets the position and $\mathbf{A}_{3\times3}$ is responsible for defining the orientation of the object. The elements of $\mathbf{q^T}$ can be controlled independently, adjusting the x, y and z coordinates of the position. Matrix elements $a_{11}, \ldots a_{33}$, however, are not independent, since the degree of freedom in orientation is 3, not 9, the number of elements in the matrix. In fact, a matrix representing a valid orientation must not change the size and the shape of the object, thus requiring the row vectors of $\mathbf{A}$ to be unit vectors forming a perpendicular triple. Matrices having this property are called **orthonormal**.

...

Concerning the interpolation, the elements of the position vector can be interpolated independently, but independent interpolation is not permitted for the elements of the orientation matrix, since the interpolated elements would make non-orthonormal matrices. A possible solution to this problem is to interpolate in the space of the roll/pitch/yaw (α, β, γ) angles (see section 5.1.1), since they form a basis in the space of the orientations, that is, any roll-pitch-yaw triple represents an orientation, and all orientations can be expressed in this way. Consequently, the time functions describing the motion are:

$$\mathbf{p}(t) = [x(t), y(t), z(t), \alpha(t), \beta(t), \gamma(t)] \tag{13.4}$$

($\mathbf{p}(t)$ is called parameter vector).

Using the definitions of the roll, pitch and yaw angles:

$$\mathbf{A} = \begin{bmatrix} \cos\alpha & \sin\alpha & 0 \\ -\sin\alpha & \cos\alpha & 0 \\ 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} \cos\beta & 0 & -\sin\beta \\ 0 & 1 & 0 \\ \sin\beta & 0 & \cos\beta \end{bmatrix} \cdot \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\gamma & \sin\gamma \\ 0 & -\sin\gamma & \cos\gamma \end{bmatrix}. \tag{13.5}$$

The position vector is obviously:

$$\mathbf{q^T} = [x, y, z]. \tag{13.6}$$

We concluded that in order to generate realistic motion, $\mathbf{T}$ should have continuous second derivatives. The interpolation is executed, however, in the space of the parameter vectors, meaning that this requirement must be replaced by another one concerning the position values and orientation angles.

The modeling transformation depends on time indirectly, through the parameter vector:

$$\mathbf{T} = \mathbf{T}(\mathbf{p}(t)). \tag{13.7}$$

Expressing the second derivative of a matrix element T_{ij}:

$$\frac{dT_{ij}}{dt} = grad_{\mathbf{p}} T_{ij} \cdot \dot{\mathbf{p}}, \tag{13.8}$$

$$\frac{d^2T_{ij}}{dt^2} = (\frac{d}{dt}\mathrm{grad}_{\mathbf{p}} T_{ij}) \cdot \dot{\mathbf{p}} + \mathrm{grad}_{\mathbf{p}} T_{ij} \cdot \ddot{\mathbf{p}} = h(\mathbf{p}, \dot{\mathbf{p}}) + H(\mathbf{p}) \cdot \ddot{\mathbf{p}}. \tag{13.9}$$

13.2 Interpolation of the camera parameters

Interpolation along the path of the camera is a little bit more difficult, since a complete camera setting contains more parameters than a position and orientation. Recall that the setting has been defined by the following independent values:

1. $v\vec{r}p$, view reference point, defining the center of the window,
2. $v\vec{p}n$, view plane normal, concerning the normal of the plane of the window,
3. $v\vec{u}p$, view up vector, showing the directions of the edges of the window,
4. $wheight, wwidth$, horizontal and vertical sizes of the window,
5. $e\vec{y}e$, the location of the camera in the u, v, w coordinate system fixed to the center of the window,
6. fp, bp, front and back clipping plane,
7. Type of projection.

Some of the above parameters are not completely independent. Vectors $v\vec{p}n$ and $v\vec{u}p$ ought to be perpendicular unit vectors. Fortunately, the algorithm generating the viewing transformation matrix $\mathbf{T_V}$ takes care of these requirements, and should the two vectors not be perpendicular or of unit length, it adjusts them. Consequently, an appropriate space of independently adjustable parameters is:

$$\mathbf{p_{cam}}(t) = [v\vec{r}p, v\vec{p}n, v\vec{u}p, wheight, wwidth, e\vec{y}e, fp, bp]. \tag{13.10}$$

As has been discussed in chapter 5 (on transformations, clipping and projection), these parameters define a viewing transformation matrix $\mathbf{T_V}$ if the following conditions hold:

$$\vec{vpn} \times \vec{vup} \neq 0, \quad wheight \geq 0, \quad wwidth \geq 0, \quad eye_w < 0, \quad eye_w < fp < bp. \tag{13.11}$$

This means that the interpolation method has to take account not only of the existence of continuous derivatives of these parameters, but also of the requirements of the above inequalities for any possible point of time. In practical cases, the above conditions are checked in the knot points (keyframes) only, and then animation is attempted. Should it be that the path fails to provide these conditions, the animation system will then require the designer to modify the keyframes accordingly.

13.3 Motion design

Define the time of knot points: $t_1, \ldots, t_n$; *// design phase*
for each knot point k **do**
  for each object o **do**
    Arrange object o:
      $\mathbf{p_o}(t_k) = [x(t_k), y(t_k), z(t_k), \alpha(t_k), \beta(t_k), \gamma(t_k)]_o$;
  endfor
  Set camera parameters: $\mathbf{p_{cam}}(t_k)$;
endfor
for each object o **do**
  Interpolate a C^2 function for:
    $\mathbf{p_o}(t) = [x(t), y(t), z(t), \alpha(t), \beta(t), \gamma(t)]_o$;
endfor
Interpolate a C^2 function for: $\mathbf{p_{cam}}(t)$;

Initialize Timer(t_{start}); *// animation phase*
do
  t = Read Timer;
  for each object o **do**
    Sample parameters of object o:
      $\mathbf{p_o} = [x(t), y(t), z(t), \alpha(t), \beta(t), \gamma(t)]_o$;
    $\mathbf{T_{M,o}} = \mathbf{T_{M,o}}(\mathbf{p_o})$;
  endfor
  Sample parameters of camera: $\mathbf{p_{cam}} = \mathbf{p_{cam}}(t)$;
  $\mathbf{T_V} = \mathbf{T_V}(\mathbf{p_{cam}})$;
  Generate Image;
while $t < t_{\text{end}}$;