Clippingul unei scene cu un volum de vedere

Pentru vizualizarea unei scene sunt necesare numai obiectele (sau părțile din obiecte) care sunt în interiorul unui **volum de vedere**. Această vizualizare face un **clipping** (decupare) a scenei la volumul de vedere.

Relația dintre un obiect din scenă şi volumul de vedere este:

- obiectul este inclus în volumul de vedere, deci proiecția pe planul de proiecție este vizibilă
- obiectul este în întregime în afara volumului de vedere, deci el poate fi ignorat în următoarele etape de la vizualizare
- obiectul este parțial inclus în volumul de vedere, deci este necesară determinarea intersecției dintre obiect şi volumul de vedere

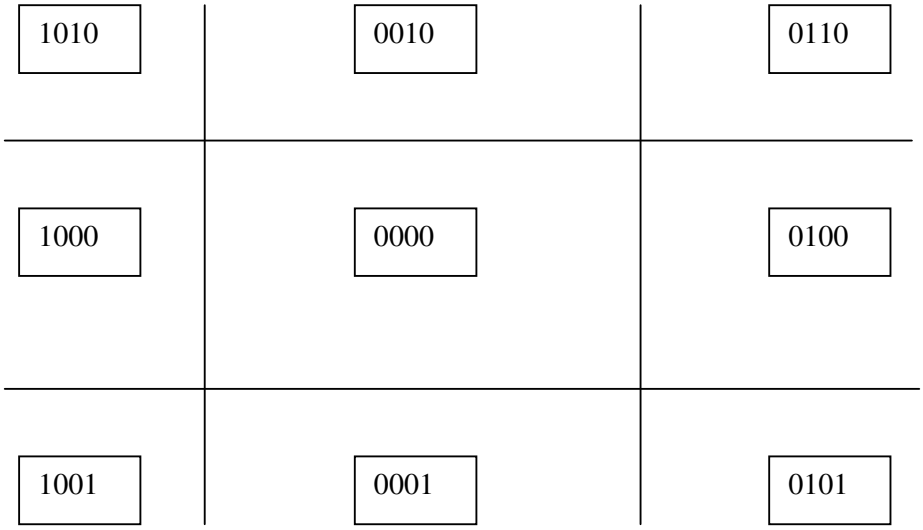
In procesul de vizualizare prin matricea de normalizare se face o transformare a volumului de vedere la domeniul: $[-1,1] \times [-1,1] \times [0,1]$. Prin această normalizare se poate determina relativ uşor relația care esistă între un obiect din scenă (care este şi el transformat prin matricea de normalizare) şi domeniul amintit. Un algoritm pentru această decupare este o generalizare a următorului algoritm ce rezolvă o problemă asemănătoare în plan.

Clippingul unui segment cu un viewport

Fie **(xmin,ymin)** şi **(xmax,ymax)** două puncte de pe ecran care definesc un **viewport** (poate fi porțiunea din planul de proiecție ce se extrage pe ecran), iar (x1,y1) şi (x2,y2) extremitățile unui segment ce trebuie desenat. Vom determina intersecția segmentului cu viewportul prin algoritmul lui **Cohen-Sutherland**.

Pentru fiecare extremitate a segmentului vom determina un **cod** (cu ajutorul funcției **CodPunct**). Acest cod precizează **poziția unui punct în raport cu viewportul**. Codul este format din patru cifre binare: $c_4c_3c_2c_1$, cu următoarea semnificație (se vede din figură):

- c_1 este 1 dacă punctul este sub viewport şi 0 în caz contrar
- c_2 este 1 dacă punctul este deasupra viewportului şi 0 în caz contrar
- c_3 este 1 dacă punctul este la dreapta viewportului şi 0 în caz contrar
- c_4 este 1 dacă punctul este la stânga viewportului şi 0 în caz contrar



```
private class punct{
   int x,y;
   public punct(int x, int y){
     this.x=x; this.y=y;
   }
}

private int CodPunct(punct p, punct p1, punct p2){
   //precizeaza pozitia punctului p
   //in raport cu viewportul dat de punctele p1 si p2
   int cod=0x00;
   if (p.x>p2.x){cod=0x04;
   }else if (p.x<p1.x) cod=0x08;
   if (p.y>p2.y){ cod=(int)(cod | 0x02);
   }else if (p.y<p1.y) cod=(int)(cod | 0x01);
   return cod;
}
```

Pentru a determina intersecția segmentului cu viewportul precizat de punctele **p1** şi **p2** se va folosi funcția **Clipping** ce va fi descrisă în continuare. Valoarea funcției va fi:

- 1 - dacă intersecția este vidă,
- 0 - dacă intersecția este nevidă, iar variabilele (punctele) **a, b** vor preciza noile extremități ale segmentului

```
private boolean Clipping(punct a, punct b, punct p1, punct p2){
  int cod1, cod2;
  cod1=CodPunct(a,p1,p2);
  cod2=CodPunct(b,p1,p2);
  if ((cod1 & cod2) !=0) return false; //clipping vid
  if ((cod1 | cod2) ==0) return true;  //ambele extr.sunt in viewport
  //cel putin o extremitate este in afara viewportului
  boolean r=true;
  PunctNou(a,b,cod1,p1,p2); cod1=CodPunct(a,p1,p2);
  PunctNou(b,a,cod2,p1,p2); cod2=CodPunct(b,p1,p2);
  if ((cod1 & cod2) !=0) r=false;
   return r;
}

private void PunctNou(punct a, punct b, int cod, punct p1, punct p2){
  if( (cod & 2) != 0){ //a este deasupra de viewportul (p1,p2)
    a.x += (b.x - a.x)*(p2.y - a.y)/(b.y - a.y);
    a.y = p2.y;
  }else
  if( (cod & 1) != 0){ //a sub viewport
    a.x += (b.x - a.x)*(p1.y - a.y)/(b.y - a.y);
    a.y = p1.y;
  }
   cod=CodPunct(a,p1,p2);//
   if( (cod & 4) != 0){ //a la dreapta de viewport
    a.y += (b.y - a.y)*(p2.x - a.x)/(b.x - a.x);
    a.x = p2.x;
  }
  else
  if( (cod & 8) != 0){ //p la stanga de viewport
```

```
    a.y += (b.y - a.y)*(p1.x - a.x)/(b.x - a.x);
    a.x = p1.x;
  }
}
```