Graphical Extensions of the *MATHEMATICA*® Package TuGames

Holger I. MEINHARDT
Institute for Statistics and
Economic Theory
University of Karlsruhe
Englerstr. 11,Bd. 11.40
D-76128 Karlsruhe
E-mails: hme@vwl3.wiwi.uni-karlsruhe.de, Holger.Meinhardt@wiwi.uni-karlsruhe.de

August 14, 2005

1 Introduction

This notebook will discuss some commands to demonstrate the graphical features of the *MATHEMATICA*® package *TuGames*. The graphical extensions are at the moment only available for UNIX/LINUX computers. Any help is welcome to port these features to Windows machines. Please consult the requirement section in the packages *IOTuGames.m*, *TuGamesView3D.m*, and *TuGamesAux.m* to get some informations what is need to run the graphical extensions of the package properly. If you should need some help during the installation procedure please do not hesitate to contact the author of the package. In the sequel we will just discuss the most important commands to derive 3D-Graphics from an underlying four person game with transferable utility (a so-called TU Game). There are additional commands available that will not be discussed here, but in general for almost all functions is a short description and explanation available that gives some informations how to use a specific function correctly. At the end of this notebook the reader will also find some references.

To start with our presentation we need to load some *MATHEMATICA*® packages. This will be done by calling

```
In[1]:= Needs["coop`CooperativeGames`"]

        Needs["TuGames`"]
        (*Needs["TuGamesAux`"]*)

        Needs["VertexEnum`"]
====================================================
Loading Package 'TuGames' for Unix
====================================================
TuGames V1.1 by Holger Meinhardt
Release Date : 27.07.2005
Program runs under MATHEMATICA® Version 3.0 or later
====================================================
====================================================
Package 'TuGames' loaded
====================================================
```

To get an overview of the basic functions and options that will be provided by the package *TuGames.m* one needs to call

```
In[2]:= ?TuGames`*
```

```
TuGames`
AdjustedEffUpperVectors   CoordS                        KernelImputationQ          SmallestContribution
AdjustedEffVector         CoreElementsQ                 ModifiedKernel             SmallestContributionVector
AdjustedWorthVectors      DefineGame                    ModifiedNucleolus          StarCriticalVal
AllConstraints            DeltaLP                       KernelSubCallEmpty         StrictlyConvexUnanConditionQ
AllMaxSurpluses           DisagreeConvex                LexiCenter                 StrIncreasMargContrib
Assign                    Disagreement                  LargestAmount              SuperAdditiveQ
AvConvexQ                 DisplayAllResults             MargValue                  TauValue
AvSumCheck                DisplayMatrixForm             MaxExcessBalanced          ThirdCriticalVal
BelongToCoreQ             DualGame                      MaxSurplus                 TIJsets
CddGmpPlotCore            EpsCore                       MinUnanimityCoordinates    Transfer
CddGmpPlotImputationSet   EpsilonValue                  MonotoneQ                  TransferConstraints
CddGmpPlotReasonableSet   EpsValue                      NewShapley                 UnanimityCoordinates
CddGmpVerticesCore        EvalSumMinCoord               Nuc1convex                 UtopiaPayoff
CddVerticesCore           ExcessPayoff                  OneNormalization           UtopiaSum
CddVerticesImputationSet  FilledCore                    PreKernelQ                 UtopiaVector
CddVerticesReasonableSet  FirstCriticalVal              Quota                      Vec3DToSimplex
ChangeInternalEps         FourthCriticalVal             RationalApproximate        VerticesCore
coal                      GameMonotoneQ                 RationalExact              WeaklySuperAdditiveQ
Concession                Gap                           ReasonableOutcome          WithIncidences
ContributionSum           ImputationQ                   ReasonableSet              ZeroMonotoneQ
ConvCheck                 IncreasingMargContributions   Scrb                       ZeroNormalization
ConvexQ                   kCover                        ScrbSolution               ZeroOneNormalization
ConvexStrQ                Kernel                        SecondCriticalVal
ConvexUnanConditionQ      KernelCalculation             SelectSuperSets
ConvStrQ                  KernelImputationListQ         SetGameToNonZeroMonotonic
```

Note that a function that gives no explanation should not be used. Such a function must be visible for other packages.

```
In[3]:= << IOTuGames`
```

```
In[4]:= << TuGamesView3D`
```

```
In[5]:= << FrontEndGraphics.m
```

Now let us first check which additional functions and options are available for the package*IOTuGames.m*.

```
In[6]:= ?IOTuGames`*
```

```
TuGames`
AnimationKernelProperty   DisplayAsMatrix            KcOutput              PlotCore3D
ShowSkeleton              SubCallStrCore             AsDigits              EpsValues
KernelCoord               SaveVertices               ShowStrongEpsCore     UpperCriticalVal
AsIncidenceMatrix         GeneratePolyhedron         LowerCriticalVal      ShapleyCoord
ShowUpperSet              VertexFacetsIncidences     CheckNucEqKernel      GenKernelGraphLine
MvOutput                  ShowCore                   SkelCore              ViewKernelSol
CriticalValue             GenKernelGraphPoint        NewDirectory          ShowKernel
SkelImputationSet         ViewNucleolusSol           defopts               GenNucleolusGraphPoint
NucleolusCoord            ShowKernelCatcher          SkelLowerSet          ViewShapleySol
defopts1                  GenShapleyGraphPoint       NumberOfVertices      ShowLowerSet
SkelReasonableSet         WithFaceNumber             defopts3              InterLockingImpReasSet
OOGLFile                  ShowNucleolus              SkelStrongEpsCore     WithProperContribution
StepSize                  InternalViewPoint          OutputDirname         ShowShapley
```

The package *TuGamesView3D.m* provides just two functions called **MvCorePlot3D[]** and **MvKernelCatcher[]**.

2 Graphical Extensions

Now, we are in the position to define a cooperative game (transferable utility game). For a more comprehensive discussion how to use the package please consult the manual file ManualTuGamesV.1.1.nb.

```
In[7]:= ExpGame := (T = {1, 2, 3, 4}; Clear[v];
          v[{}] = 0;
          v[{1}] = 0;
          v[{2}] = 0;
          v[{3}] = 0;
          v[{4}] = 0;
          v[{1, 2}] = 0;
          v[{1, 3}] = 1/4;
          v[{1, 4}] = 2/4;
          v[{2, 3}] = 1/4;
          v[{2, 4}] = 3/4;
          v[{3, 4}] = 0;

          v[{1, 2, 3}] = 1;
          v[{1, 2, 4}] = 1;
          v[{1, 3, 4}] = 1;
          v[{2, 3, 4}] = 1; v[T] = 2; )
```

To produce some 3D graphics we have first to check some game properties. Especially, we should verify if the core solution is non-empty. Moreover, we check if some additional game properties are satisfied. For instance, by checking the convexity property we can deduce if the kernel solution is a singleton and coincides with the nucleolus of the game.

```
In[8]:= ConvexQ[ExpGame]
Out[8]= False

In[9]:= AvConvexQ[ExpGame]
```

```
Out[9]= True

In[10]:= CoreQ[ExpGame]
Out[10]= True

In[11]:= ZeroMonotoneQ[ExpGame]
Out[11]= True
```

In the next step we compute some solutions that will be used later on to produce the 3D graphics. In the package *TuGames.m* different algorithm for computing, for instance, the kernel, nucleolus and the Shapley value are implemented.

```
In[12]:= gsh1 = ShapleyValue[ExpGame]
```
Out[12]= $\{\frac{23}{48}, \frac{25}{48}, \frac{7}{16}, \frac{9}{16}\}$

```
In[13]:= sh1 = NewShapley[ExpGame]
```
Out[13]= $\{\frac{23}{48}, \frac{25}{48}, \frac{7}{16}, \frac{9}{16}\}$

```
In[14]:= nuc1 = Nucleolus[ExpGame]
```
Out[14]= $\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}$

```
In[15]:= ModifiedNucleolus[ExpGame]
```
Out[15]= $\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}$

```
In[16]:= LexiCenter[ExpGame]
```
Out[16]= $\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}$

```
In[17]:= ker1 = Kernel[ExpGame]
Game has nonempty core
```
Out[17]= $\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}$

```
In[18]:= ModifiedKernel[ExpGame]
```
Out[18]= $\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}$

Due to the fact that the game is not convex but average-convex, we are searching for additional kernel solutions. This can be done by calling

In[19]:= **ker11 = Kernel[ExpGame, EpsilonValue →** $\frac{1}{2}$**]**

Out[19]= $\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}$

In[20]:= **ker12 = Kernel[ExpGame, EpsilonValue →** $\frac{1}{4}$**]**

Out[20]= $\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}$

```
In[21]:= {time13, {ker13, pay13}} = AbsoluteTiming[KernelCalculation[ExpGame]]
Game is average - convex? True
```
A Kernel solution is : $\{x[1] \to \frac{5}{12}, x[2] \to \frac{7}{12}, x[3] \to \frac{5}{12}, x[4] \to \frac{7}{12}\}$

Out[21]= {2.57008 Second,

$\{\{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\}, \{\{\frac{1}{4}, \frac{1}{2}, \frac{1}{2}, \frac{3}{4}\}, \{\frac{3}{8}, \frac{5}{8}, \frac{7}{16}, \frac{9}{16}\}, \{\frac{5}{12}, \frac{7}{12}, \frac{5}{12}, \frac{7}{12}\},$
$\{\frac{1}{2}, \frac{1}{4}, \frac{1}{2}, \frac{3}{4}\}, \{\frac{1}{2}, \frac{1}{2}, \frac{1}{4}, \frac{3}{4}\}, \{\frac{1}{2}, \frac{3}{4}, \frac{1}{4}, \frac{1}{2}\}, \{\frac{3}{4}, \frac{1}{2}, \frac{1}{4}, \frac{1}{2}\}\}\}\}$

```
In[22]:= KernelImputationListQ[ExpGame, pay13]
Out[22]= {False, False, True, False, False, False, False}

In[23]:= PreKernelQ[ExpGame, pay13]
Out[23]= {False, False, True, False, False, False, False}
```

For a four person game the maximal dimension of the kernel is one and the geometrical structure is either a line segment or an unique point, we can try to find this line segment by using the function

```
In[24]:= KernelVertices[ExpGame]
Game has nonempty core
Out[24]= {{5/12, 7/12, 5/12, 7/12}}
```

```
In[25]:= {vert13, inc13} = CddGmpVerticesCore[ExpGame, WithIncidences → True]
Out[25]= {{{0, 0, 1, 1}, {0, 1, 1/2, 1/2}, {0, 1, 1/4, 3/4}, {0, 1/2, 1, 1/2}, {0, 3/4, 1/4, 1}, {1, 0, 1/4, 3/4},
           {1, 1, 0, 0}, {1, 1/4, 0, 3/4}, {1, 3/4, 1/4, 0}, {1/2, 1, 1/2, 0}, {1/2, 3/4, 3/4, 0}, {1/4, 0, 1, 3/4},
           {1/4, 1, 0, 3/4}, {1/4, 1/2, 1, 1/4}, {1/4, 3/4, 0, 1}, {3/4, 0, 1/4, 1}, {3/4, 1/4, 0, 1}},
         {{1, 2, 5, 11, 12, 15}, {2, 8, 9, 14, 15}, {2, 8, 11, 15}, {2, 9, 12, 15},
           {3, 6, 13, 15}, {3, 6, 11, 15}, {3, 8, 11, 15}, {3, 8, 14, 15}, {7, 9, 12, 15},
           {4, 7, 9, 15}, {4, 9, 14, 15}, {4, 7, 13, 15}, {3, 4, 10, 13, 14, 15}, {1, 7, 12, 15},
           {1, 7, 13, 15}, {1, 6, 13, 15}, {1, 6, 11, 15}, {1, 3, 4, 6, 7, 10, 13, 16},
           {2, 3, 4, 8, 9, 10, 14, 16}, {1, 2, 3, 5, 6, 8, 11, 16}, {1, 2, 4, 5, 7, 9, 12, 16}}}
```

```
In[26]:= KernelImputationListQ[ExpGame, vert13]
Out[26]= {False, False, False, False, False, False, False, False,
          False, False, False, False, False, False, False, False, False}
```

```
In[27]:= BelongToCoreQ[ExpGame, vert13]
Out[27]= {True, True, True, True, True, True, True,
          True, True, True, True, True, True, True, True, True, True}
```

```
In[28]:= dir1 = "transfer/wolfram/tugames/Documentation/English"
Out[28]= transfer/wolfram/tugames/Documentation/English
```

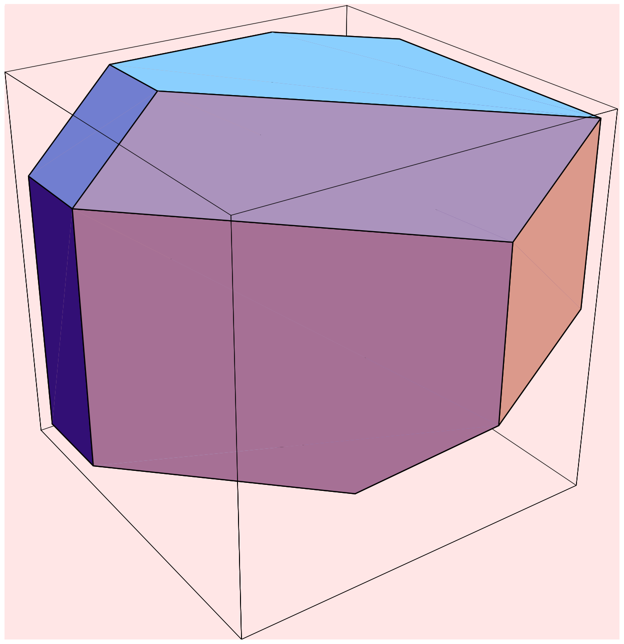
A first simple graphic can be derived with the function **PlotCore3D[].** All options can be called by using

```
In[29]:= Options[PlotCore3D]
Out[29]= {AmbientLight → GrayLevel[0], AspectRatio → Automatic, Axes → False, AxesEdge → Automatic,
          AxesLabel → None, AxesStyle → Automatic, Background → Automatic, Boxed → True,
          BoxRatios → Automatic, BoxStyle → Automatic, ColorOutput → Automatic, DefaultColor → Automatic,
          Epilog → {}, FaceGrids → None, ImageSize → Automatic, InternalViewPoint → True,
          KernelCoord → {}, Lighting → True, LightSources → {{{1., 0., 1.}, RGBColor[1, 0, 0]},
              {{1., 1., 1.}, RGBColor[0, 1, 0]}, {{0., 1., 1.}, RGBColor[0, 0, 1]}},
          NucleolusCoord → {}, OOGLFile → False, OutputDirname → False, Plot3Matrix → Automatic,
          PlotLabel → None, PlotRange → Automatic, PlotRegion → Automatic,
          PolygonIntersections → True, Prolog → {}, RenderAll → True, Shading → True,
          ShapleyCoord → {}, SphericalRegion → False, Ticks → Automatic, ViewCenter → Automatic,
          ViewKernelSol → False, ViewNucleolusSol → False, ViewPoint → {1.714, 2.478, 1.54},
          ViewShapleySol → False, ViewVertical → {0., 0., 1.}, DefaultFont :→ $DefaultFont,
          DisplayFunction :→ $DisplayFunction, FormatType :→ $FormatType, TextStyle :→ $TextStyle}
```

Note that the graphical functions that will be discussed in the sequel, need at least three arguments. A working directory must be specified, in our example this was done by defining the above argument **dir1.** The working directory is needed to write an auxiliary file on the harddisk, then the filename of this auxiliary file must be given, and the name of the game. In addition some options can be specified.

The command **PlotCore3D[]** plots the core solution as simple 3D-graphic, but not in the correct orientation.

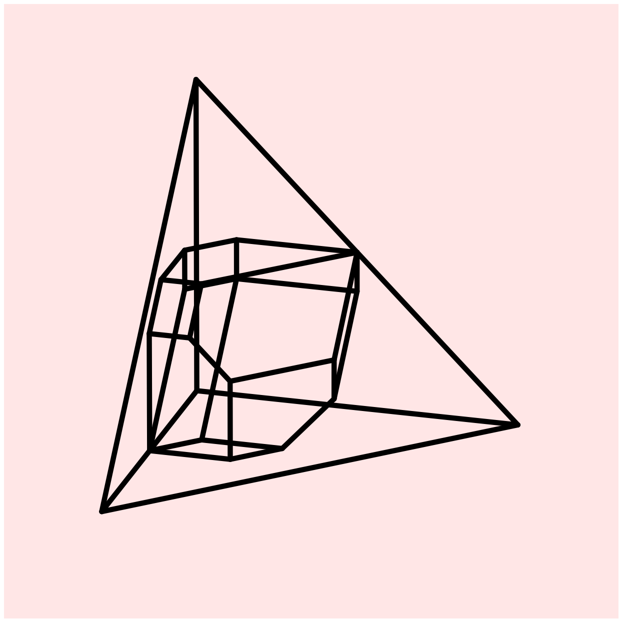
```
In[30]:= PlotCore3D[dir1, "ExpGameCore01", ExpGame, OOGLFile → False];
```



```
In[31]:= Options[SkelCore]
Out[31]= {AmbientLight → GrayLevel[0], AspectRatio → Automatic,
     Axes → False, AxesEdge → Automatic, AxesLabel → None, AxesStyle → Automatic,
     Background → Automatic, Boxed → True, BoxRatios → Automatic, BoxStyle → Automatic,
     ColorOutput → Automatic, DefaultColor → Automatic, Epilog → {}, FaceGrids → None,
     ImageSize → Automatic, InternalViewPoint → True, KernelCoord → {}, Lighting → True,
     LightSources → {{{1., 0., 1.}, RGBColor[1, 0, 0]}, {{1., 1., 1.}, RGBColor[0, 1, 0]},
        {{0., 1., 1.}, RGBColor[0, 0, 1]}}, MvOutput → False, NucleolusCoord → {},
     OOGLFile → False, OutputDirname → False, Plot3Matrix → Automatic, PlotLabel → None,
     PlotRange → Automatic, PlotRegion → Automatic, PolygonIntersections → True,
     Prolog → {}, RenderAll → True, Shading → True, ShapleyCoord → {}, ShowSkeleton → True,
     SphericalRegion → False, Ticks → Automatic, ViewCenter → Automatic, ViewKernelSol → False,
     ViewNucleolusSol → False, ViewPoint → {1.714, 2.478, 1.54}, ViewShapleySol → False,
     ViewVertical → {0., 0., 1.}, WithProperContribution → False, DefaultFont :→ $DefaultFont,
     DisplayFunction :→ $DisplayFunction, FormatType :→ $FormatType, TextStyle :→ $TextStyle}
```

A raw core skeleton interlocked with the imputation set can be obtained with the command **SkelCore[]**.

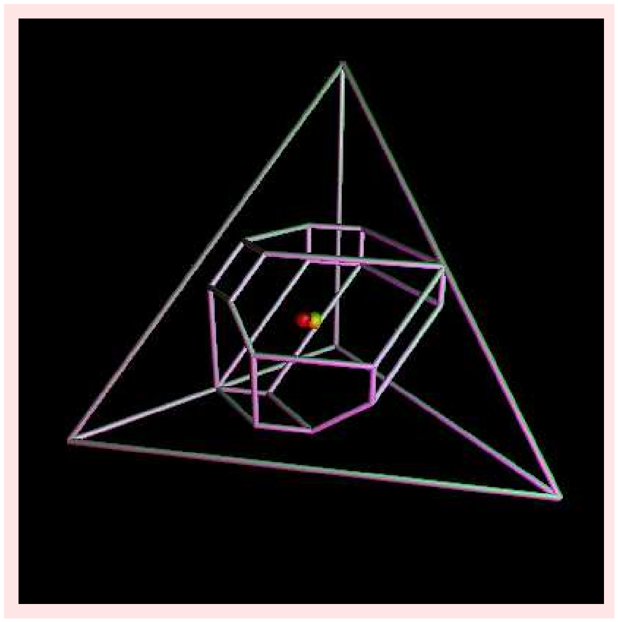
```
In[32]:= SkelCore[dir1, "ExpGameCore01", ExpGame];
```



```
In[33]:= Options[MvCorePlot3D]
Out[33]= {Boxed → False, CriticalValue → {1}, InternalViewPoint → True, KernelCoord → {},
          MVAlpha → 1., MVGrayBackground → False, MVLineTubeSize → 0.01, MVMaxVertexNumber → 4,
          MVNewScene → False, MVPointSphereSize → 0.01, MVPolygonShading → MVSmooth, MVTubeAngle → 30,
          MVTubeSegments → 12, NucleolusCoord → {}, OutputDirname → False, SaveAsEPS → False,
          SaveAsPOV → False, ShapleyCoord → {}, ShowCore → True, ShowStrongEpsCore → False,
          ViewKernelSol → False, ViewNucleolusSol → False, ViewShapleySol → False}
```

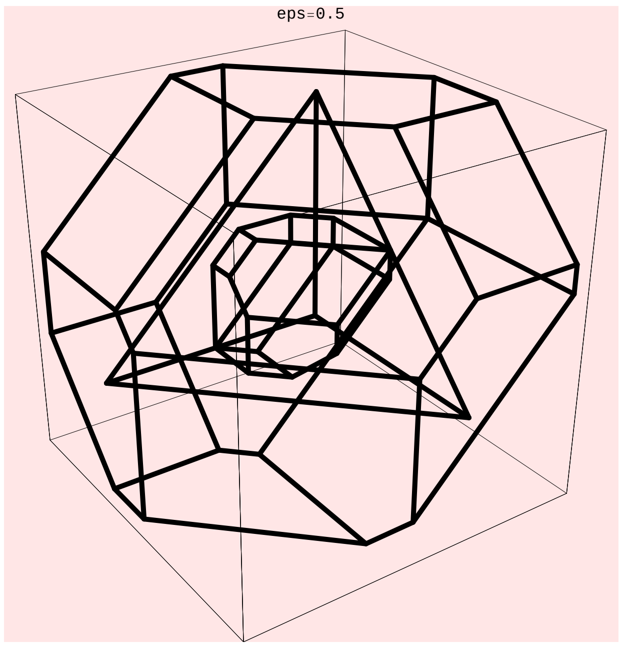
A more sophisticated graphic can be obtained by using the functions **MvCorePlot3D[]** and **MvKernelCatcher[]**. We start with the presentation of the command **MvCorePlot3D[]**.

```
In[34]:= MvCorePlot3D[dir1, "ExpGameCore02", ExpGame, ShowStrongEpsCore → False, ShowCore → True,
          ViewPoint- > {1.714, 2.478, 1.54}, ViewKernelSol → True, KernelCoord → ker1,
          ViewNucleolus → True, NucleolusCoord → nuc, ViewShapleySol → True, ShapleyCoord → sh1]
```



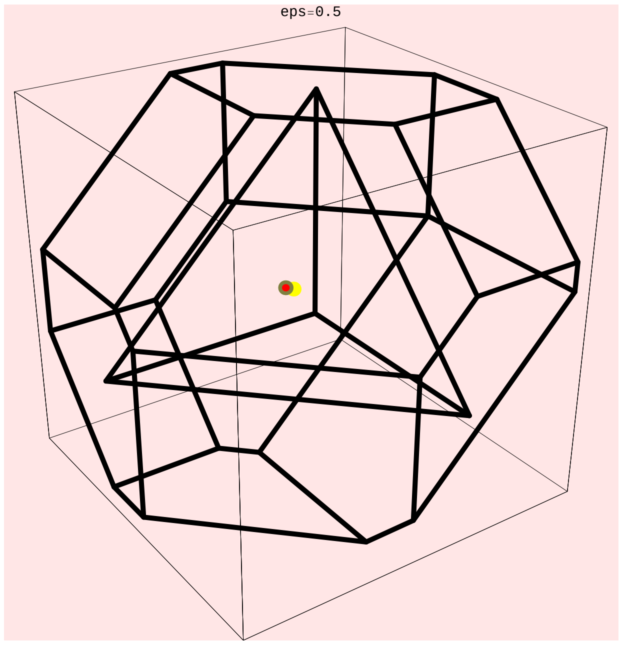
```
In[35]:= Options[SkelStrongEpsCore]
Out[35]= {AmbientLight → GrayLevel[0], AspectRatio → Automatic, Axes → False,
          AxesEdge → Automatic, AxesLabel → None, AxesStyle → Automatic, Background → Automatic,
          Boxed → True, BoxRatios → Automatic, BoxStyle → Automatic, ColorOutput → Automatic,
          CriticalValue → {1}, DefaultColor → Automatic, Epilog → {}, EpsValues → {},
          FaceGrids → None, ImageSize → Automatic, InternalViewPoint → True, KcOutput → False,
          KernelCoord → {}, Lighting → True, LightSources → {{{1., 0., 1.}, RGBColor[1, 0, 0]},
             {{1., 1., 1.}, RGBColor[0, 1, 0]}, {{0., 1., 1.}, RGBColor[0, 0, 1]}},
          MvOutput → False, NucleolusCoord → {}, OOGLFile → False, OutputDirname → False,
          Plot3Matrix → Automatic, PlotLabel → None, PlotRange → Automatic, PlotRegion → Automatic,
          PolygonIntersections → True, Prolog → {}, RenderAll → True, Shading → True,
          ShapleyCoord → {}, ShowCore → False, ShowSkeleton → True, ShowStrongEpsCore → True,
          SphericalRegion → False, Ticks → Automatic, ViewCenter → Automatic, ViewKernelSol → False,
          ViewNucleolusSol → False, ViewPoint → {1.714, 2.478, 1.54}, ViewShapleySol → False,
          ViewVertical → {0., 0., 1.}, WithProperContribution → False, DefaultFont :→ $DefaultFont,
          DisplayFunction :→ $DisplayFunction, FormatType :→ $FormatType, TextStyle :→ $TextStyle}

In[36]:= AbsoluteTiming[SkelStrongEpsCore[dir1, "ExpGameStrCore02", ExpGame, InternalViewPoint → True,
           ShowCore → True, ViewPoint- > {1.714, 2.478, 1.54}, ViewKernelSol → False,
           KernelCoord → ker1, ViewNucleolusSol → False, NucleolusCoord → nuc1,
           ViewShapleySol → False, ShapleyCoord → sh1]]
```



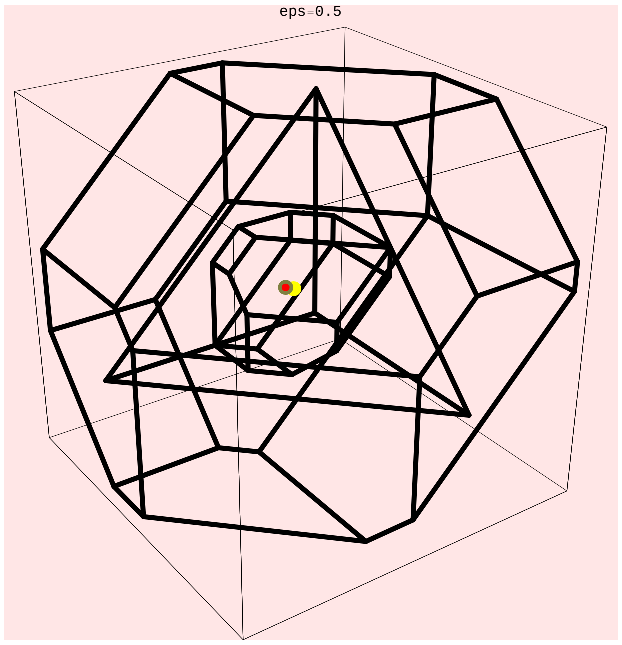
```
Out[36]= {1.00706 Second, -Graphics3D-}

In[37]:= AbsoluteTiming[SkelStrongEpsCore[dir1, "ExpGameStrCore03", ExpGame,
           InternalViewPoint → False, ShowCore → False, ViewPoint- > {1.714, 2.478, 1.54},
           ViewKernelSol → True, KernelCoord → {ker1, ker12}, ViewNucleolusSol → True,
           NucleolusCoord → nuc1, ViewShapleySol → True, ShapleyCoord → sh1]]
```



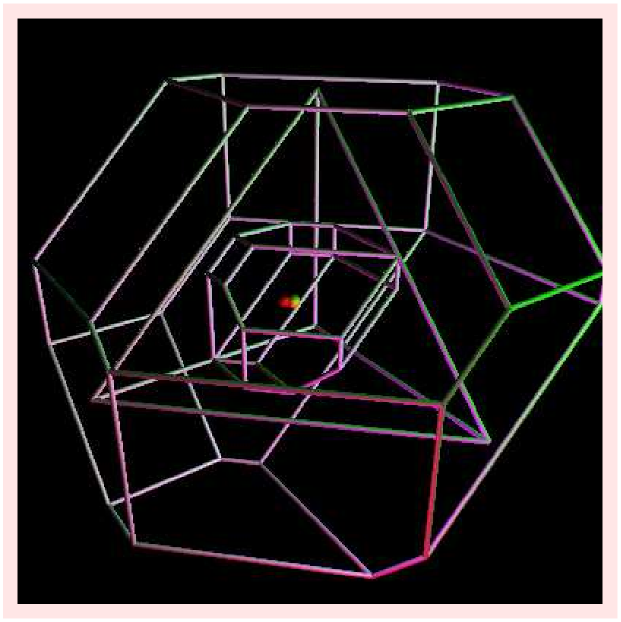
```
Out[37]= {1.0273 Second, -Graphics3D-}

In[38]:= AbsoluteTiming[SkelStrongEpsCore[dir1, "ExpGameStrCore03", ExpGame,
           InternalViewPoint → False, ShowCore → True, ViewPoint- > {1.714, 2.478, 1.54},
           ViewKernelSol → True, KernelCoord → {ker1, ker12}, ViewNucleolusSol → True,
           NucleolusCoord → nuc1, ViewShapleySol → True, ShapleyCoord → sh1]]
```



```
Out[38]= {1.06935 Second, -Graphics3D-}

In[39]:= AbsoluteTiming[MvCorePlot3D[dir1, "ExpGameCore02", ExpGame, ShowStrongEpsCore → True,
           ShowCore → True, ViewPoint- > {1.714, 2.478, 1.54}, ViewKernelSol → True,
           KernelCoord → ker1, ViewNucleolus → True, NucleolusCoord → nuc, ViewShapleySol → True,
           ShapleyCoord → sh1]]
```



```
Out[39]= {7.48275 Second, Null}
```

In the next step we want to vary the strong epsilon-core to demonstrate a crucial property of the kernel solution. The strong epsilon-core can be regarded as a window that will expose the complete kernel solution for sufficiently large epsilon values. Moreover, each non-empty strong epsilon-core has an non-empty intersection with the kernel solution that bisects $n(n-1)/2$ line segments in the strong epsilon-core with end-points in the strong epsilon-core, where n is the number of players in the game. The package *TuGames* provides several functions to compute different critical epsilon values which give in general a very crude approximation on the epsilon value to guarantee that the entire kernel solution is contained in the strong epsilon-core. Due to the crude approximation given by one of the critical values below, we know that the complete kernel solution is contained in the strong epsilon-core when the epsilon value is greater than or equal to zero, that is, the entire kernel solution is contained in the core of the game. This critical value can be computed by the function **ThirdStarCriticalVal[]**. Moreover, the function **FirstCriticalVal[]** computes the epsilon value for the least-core, that is, the smallest non-empty strong epsilon-core (see Maschler, Peleg and Shapley (1979)). The epsilon value for the least-core is attained for our example at a value of (-5/12), thus, that is the threshold value at which the strong epsilon-core will be vanish.

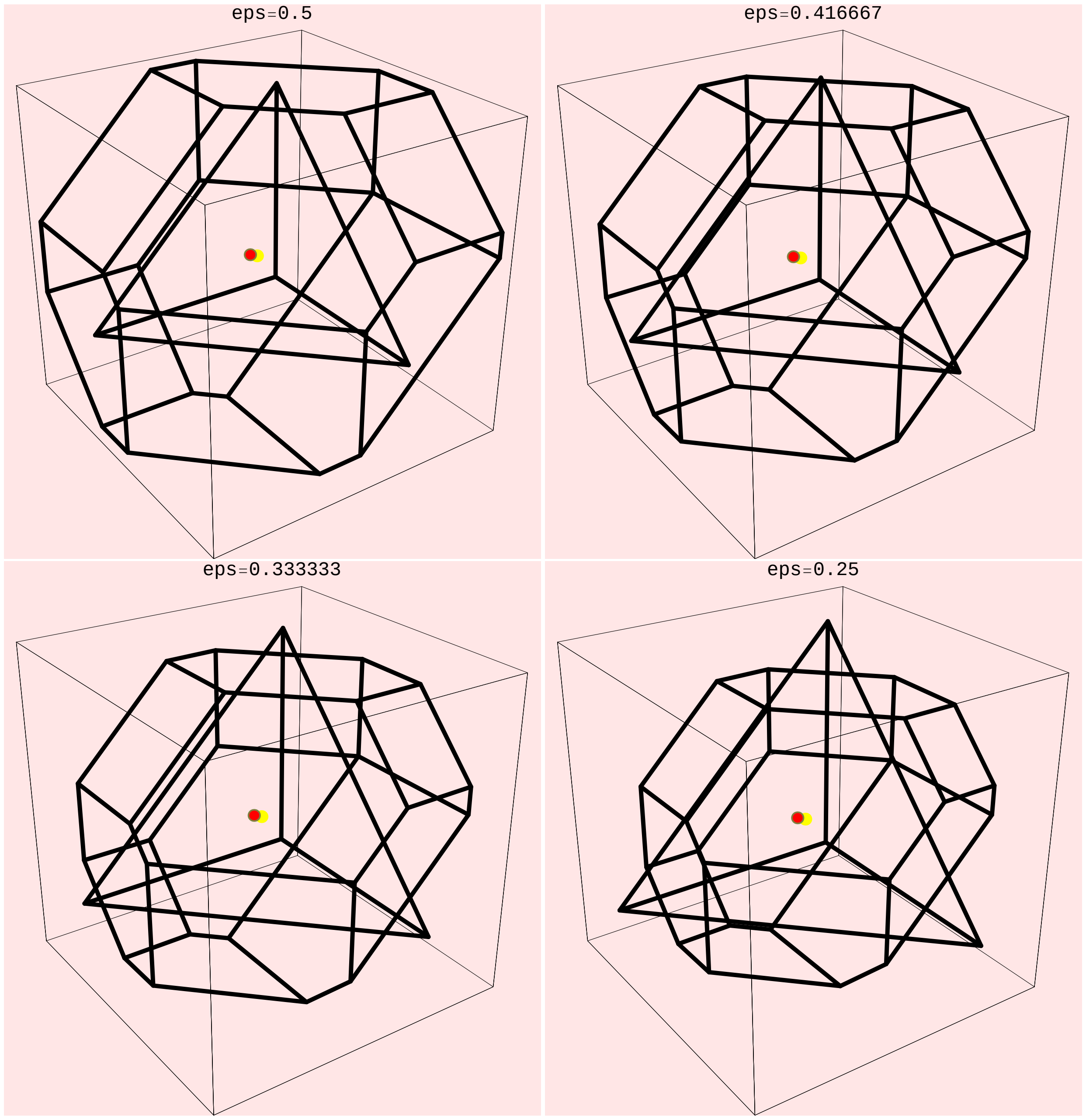
```
In[40]:= FirstCriticalVal[ExpGame]
```

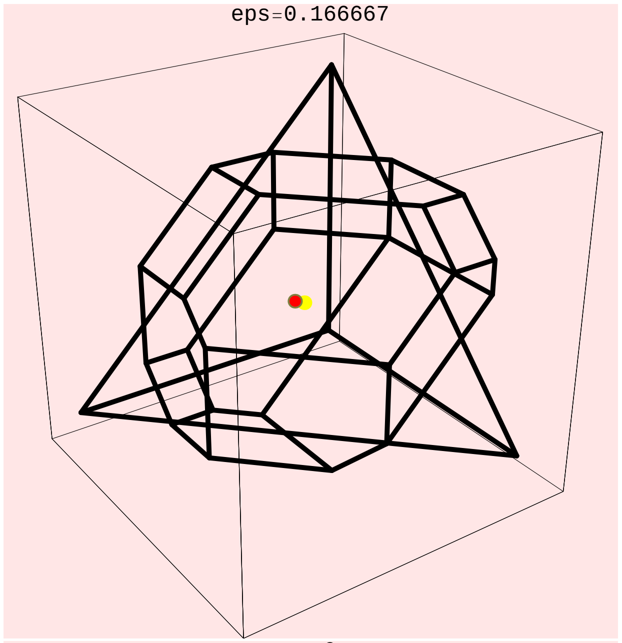
Out[40]= $\{\text{eps1} \to -\frac{5}{12}\}$

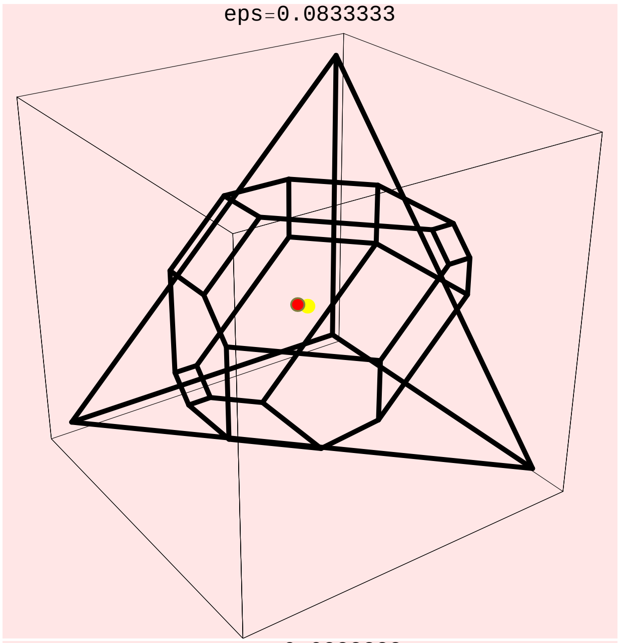
```
In[41]:= ThirdStarCriticalVal[ExpGame]
Out[41]= {thstareps → 0}
```

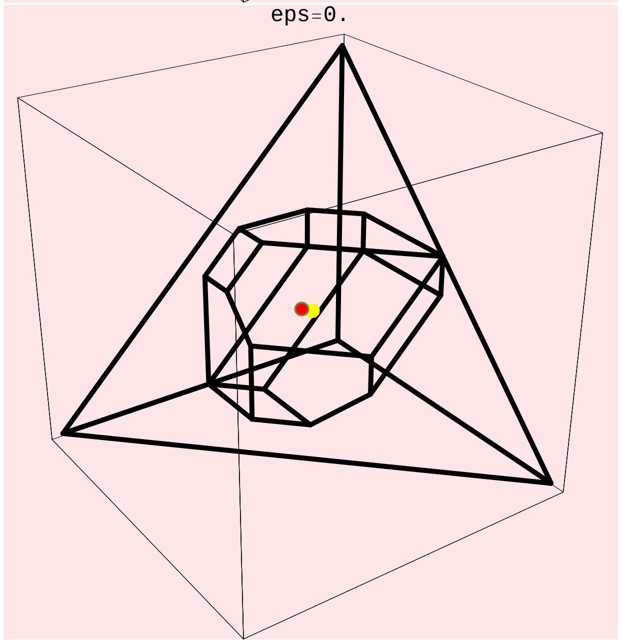
The next two commands demonstrate how one could use *MATHEMATICA*® to visualize the geometrical properties of the kernel solution. The epsilon-core will be varied from an epsilon value of (1/2), respectively from (3/4), up to -(1/4) by using a step size of -(1/12), respectively -(1/24). All values are zero-one normalized so the upper critical value should not be chosen larger than or equal to one, otherwise the strong epsilon-core might not be plotted correctly.

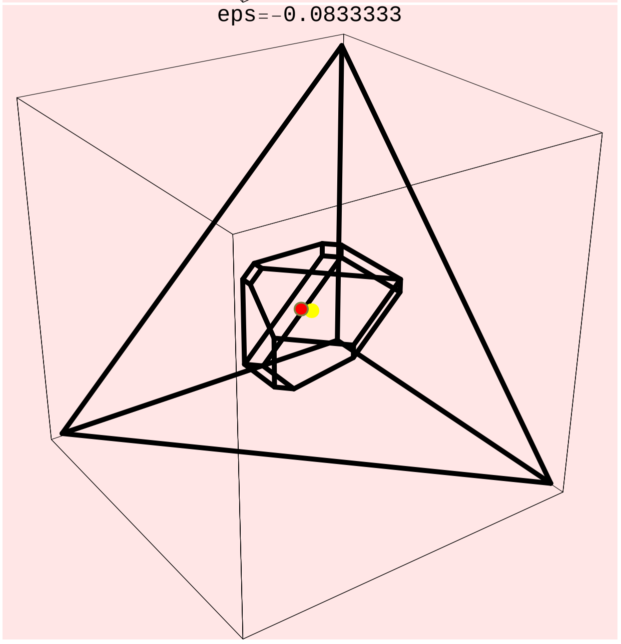
```
In[42]:= Do[SkelStrongEpsCore[dir1, "ExpGameStrCore02", ExpGame, EpsValues → {t},
           ShowCore → False, ViewPoint- > {1.714, 2.478, 1.54}, ViewKernelSol → True,
           KernelCoord → {ker1, ker12}, ViewNucleolusSol → True, NucleolusCoord → nuc1,
           ViewShapleySol → True, ShapleyCoord → sh1], {t, 1/2, -1/4, -1/12}]
```

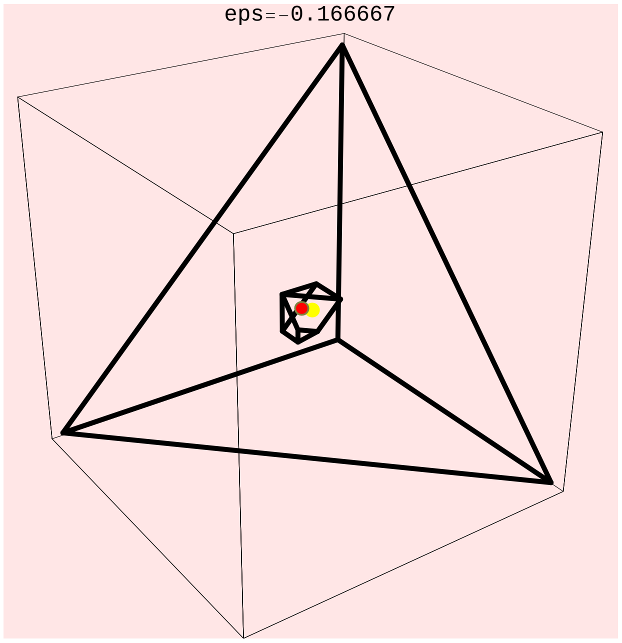

eps=0.5
eps=0.416667
eps=0.333333
eps=0.25

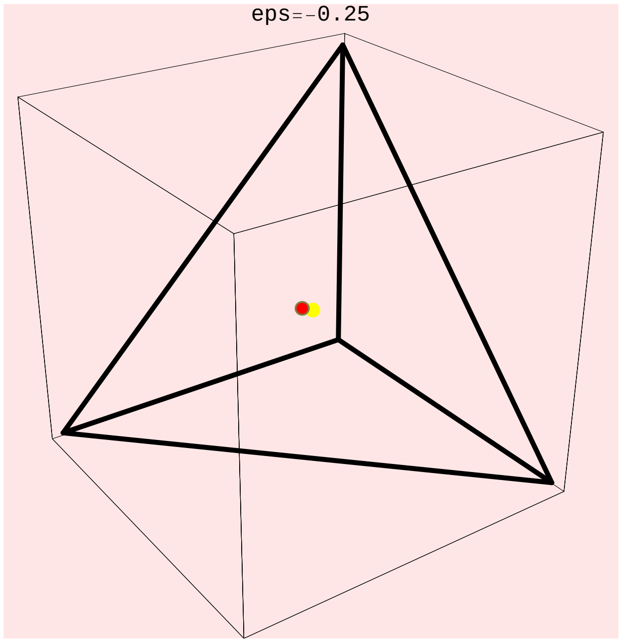

eps=0.166667


eps=0.0833333


eps=0.

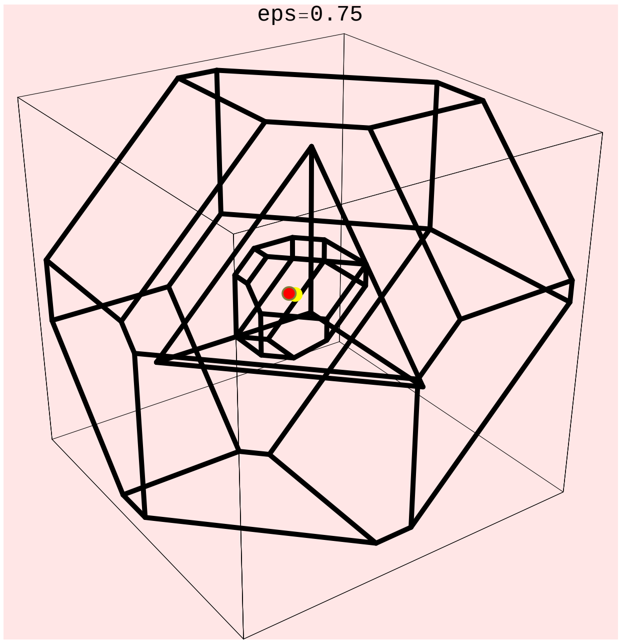

eps=−0.0833333

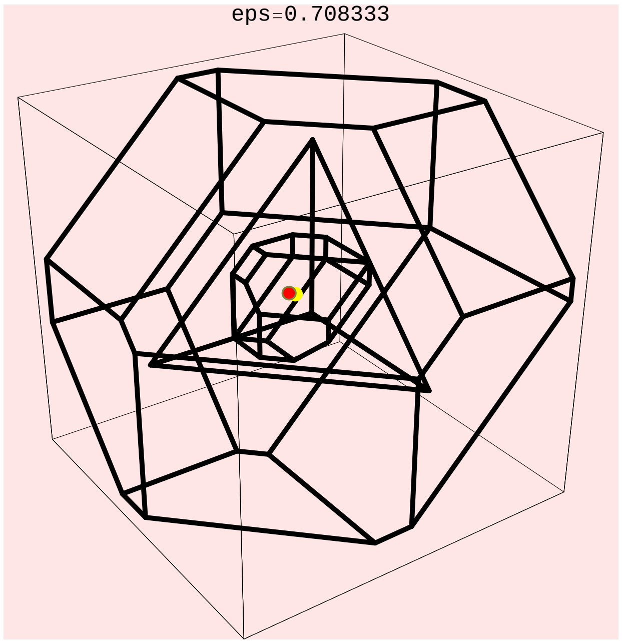


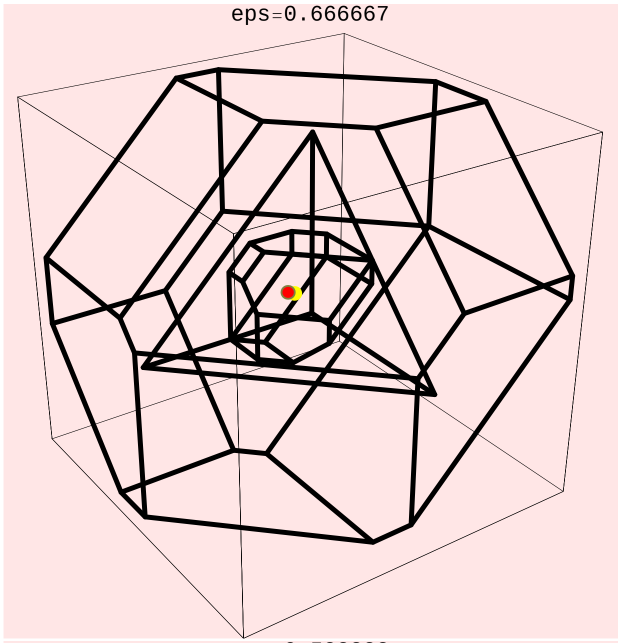


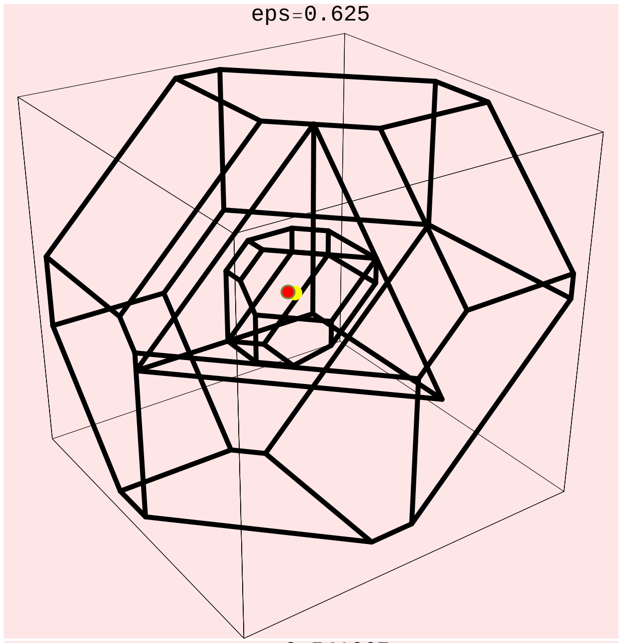
The same result can be obtained by calling the function

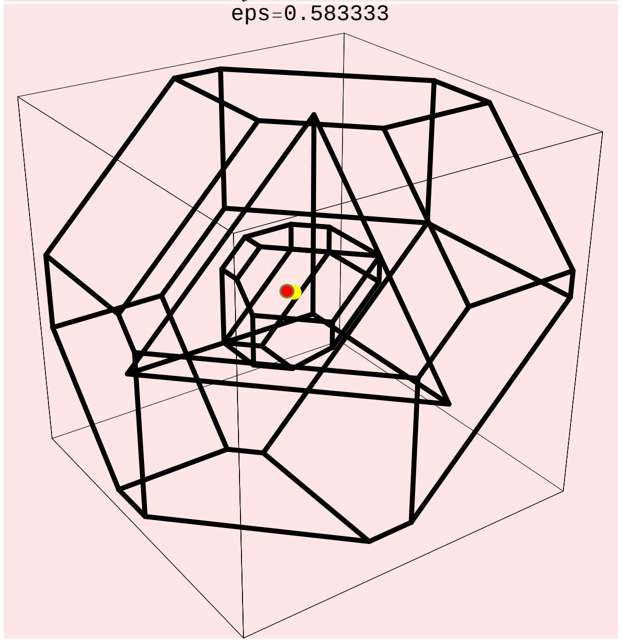
```
In[43]:= AnimationKernelProperty[dir1, "ExpGameStrCore02", ExpGame, ShowCore → True,
    ViewPoint- > {1.714, 2.478, 1.54}, ViewKernelSol → True, KernelCoord → {ker1, ker12},
    ViewNucleolusSol → True, NucleolusCoord → nuc1, ViewShapleySol → True, ShapleyCoord → sh1,
    UpperCriticalVal → {3/4}, LowerCriticalVal → {-1/4}, StepSize → {-1/24}]
```

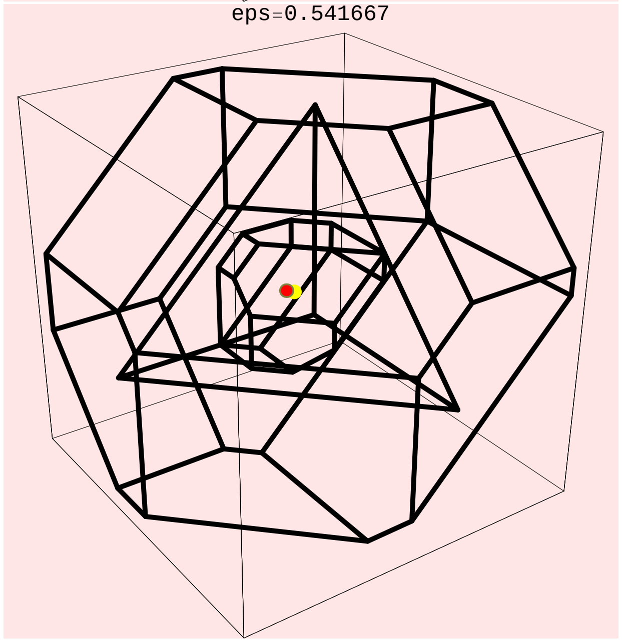


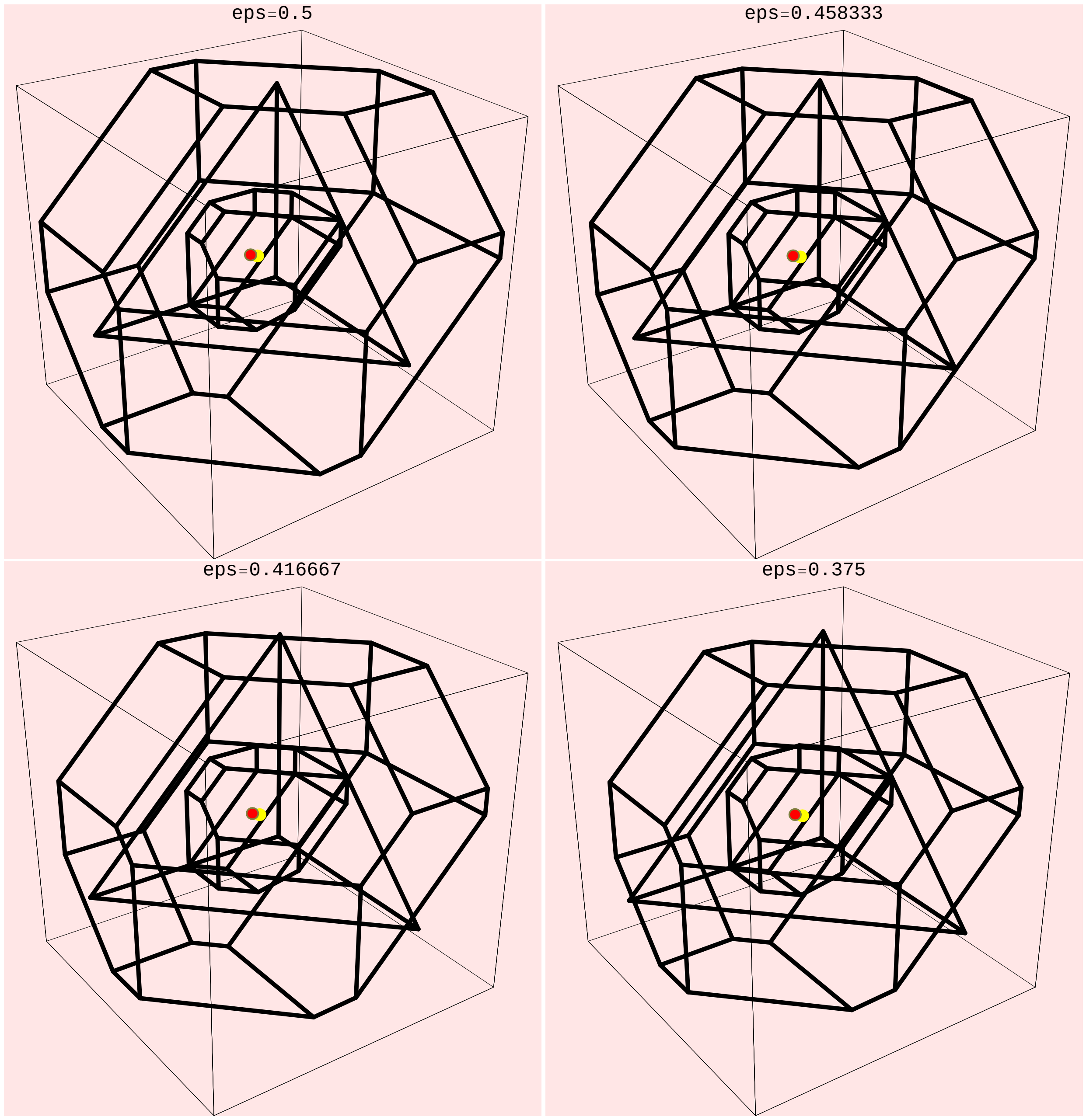


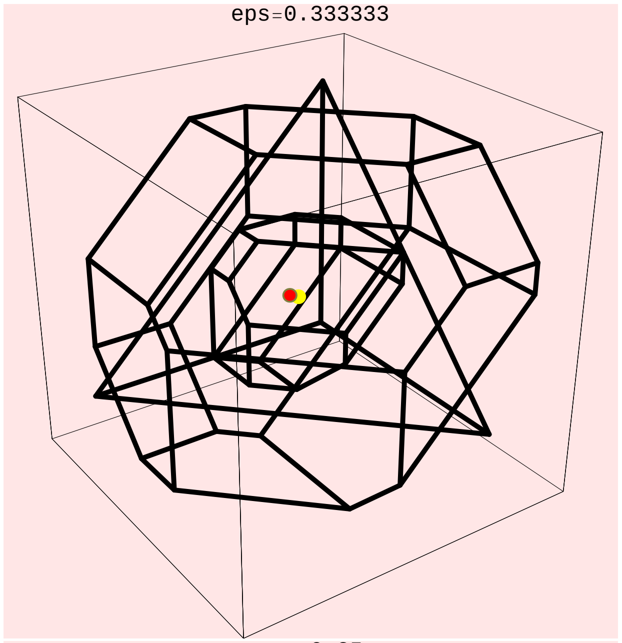

eps=0.666667

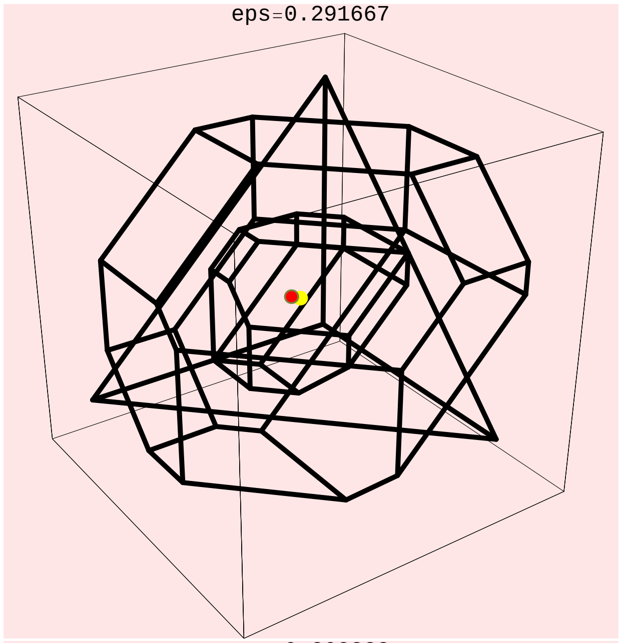

eps=0.625

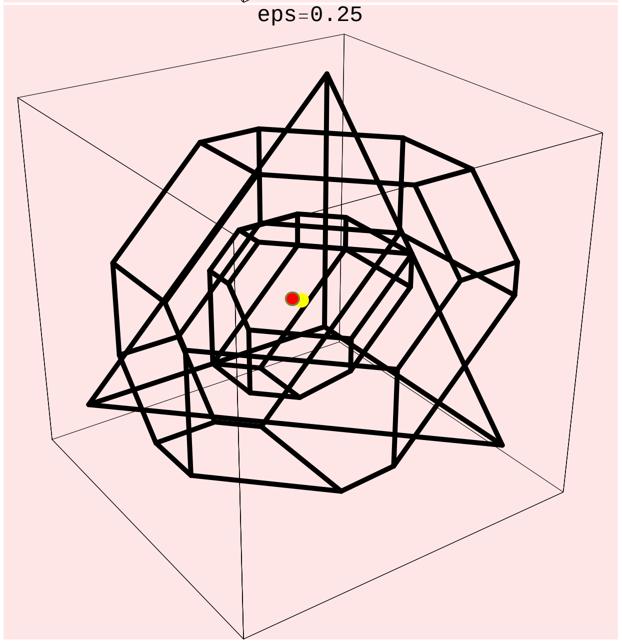

eps=0.583333

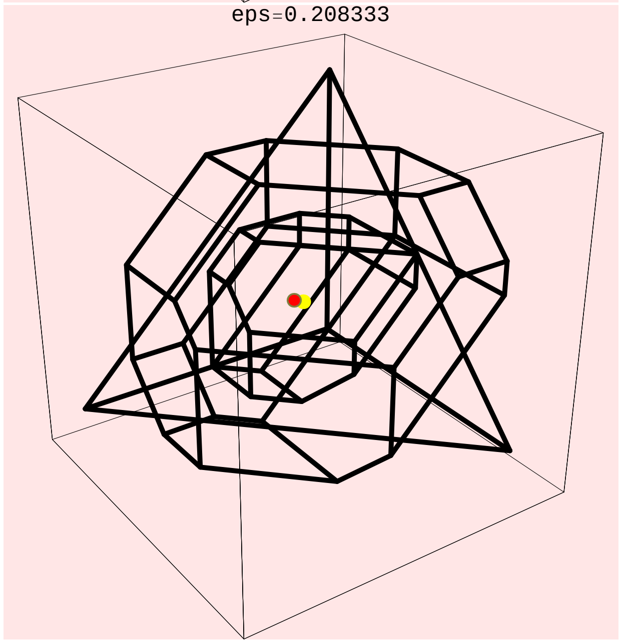

eps=0.541667

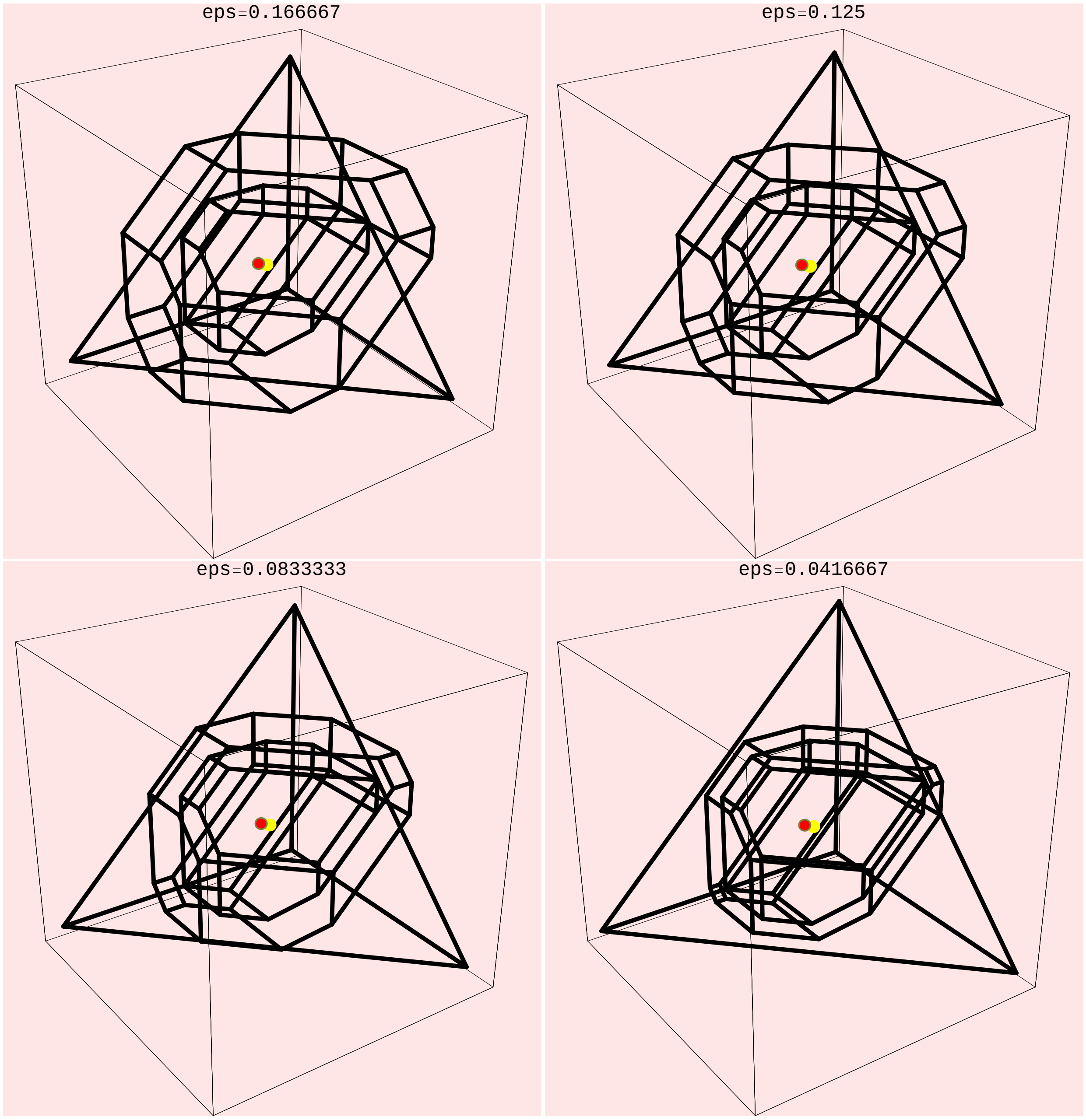

eps=0.5
eps=0.458333
eps=0.416667
eps=0.375

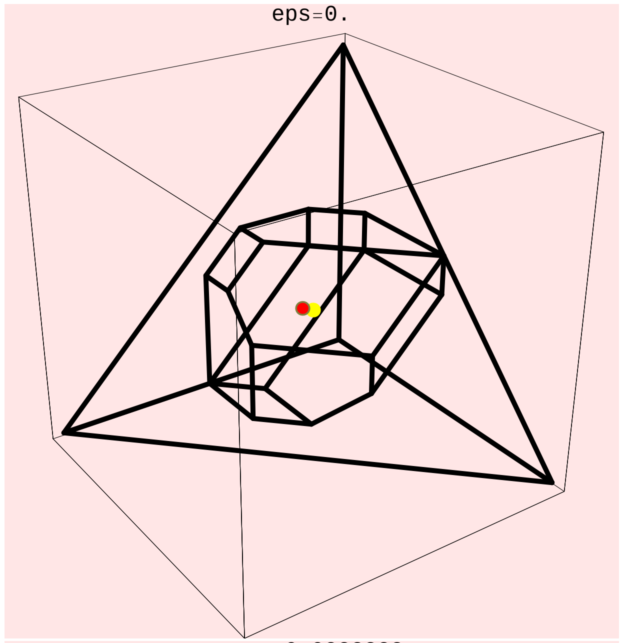

eps=0.333333

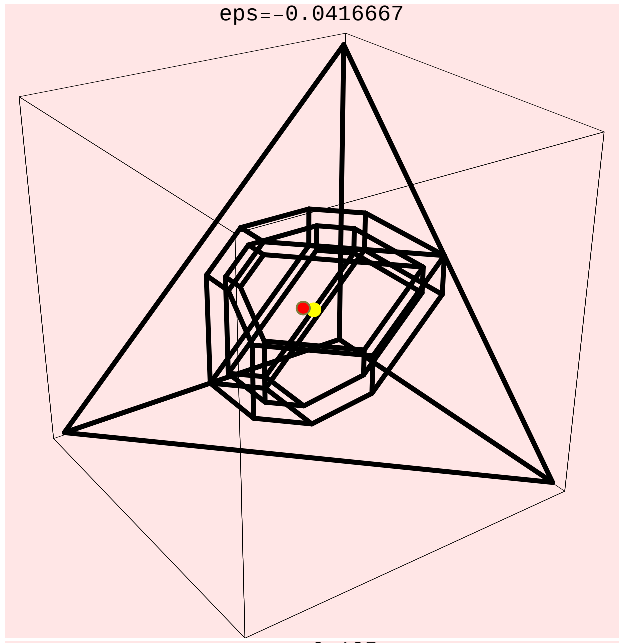

eps=0.291667

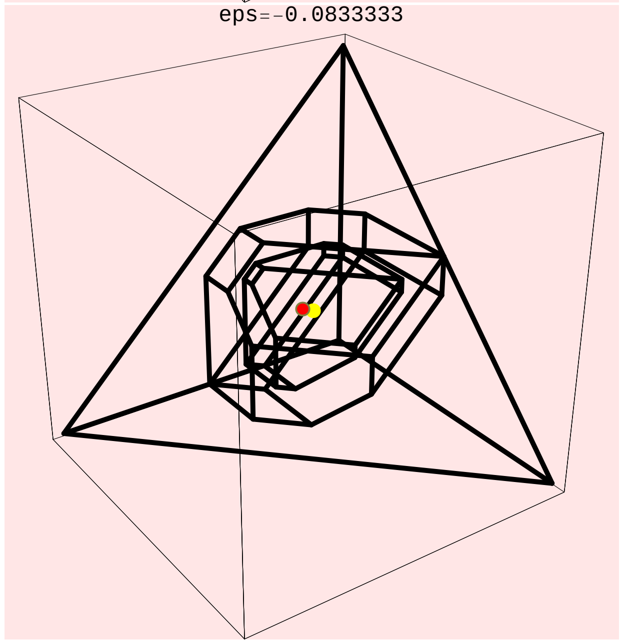

eps=0.25

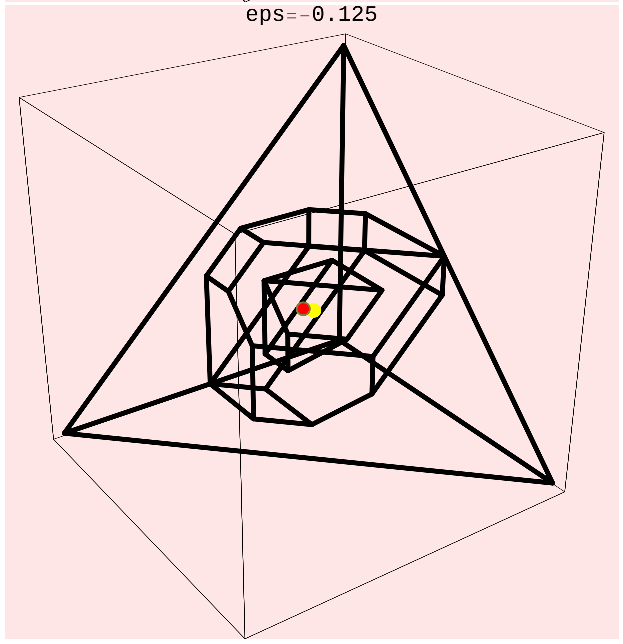

eps=0.208333

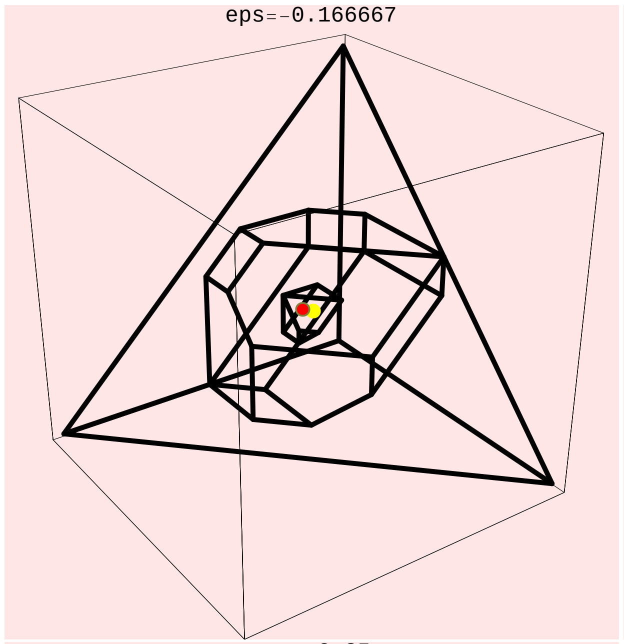

eps=0.166667
eps=0.125
eps=0.0833333
eps=0.0416667

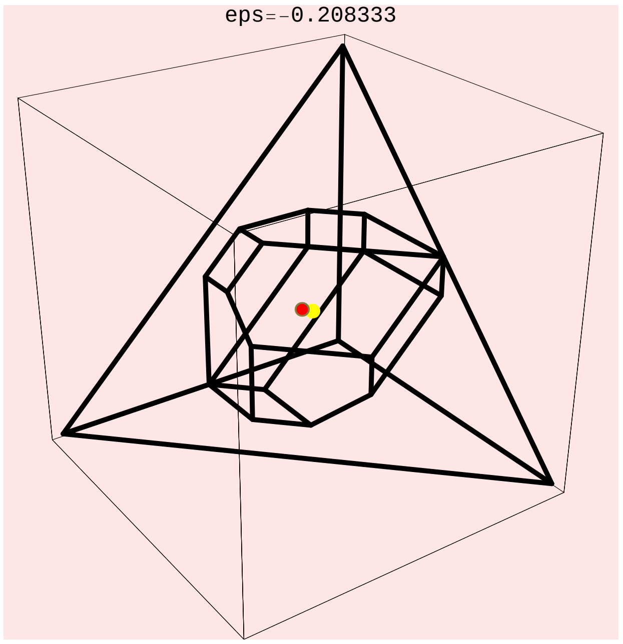

eps=0.

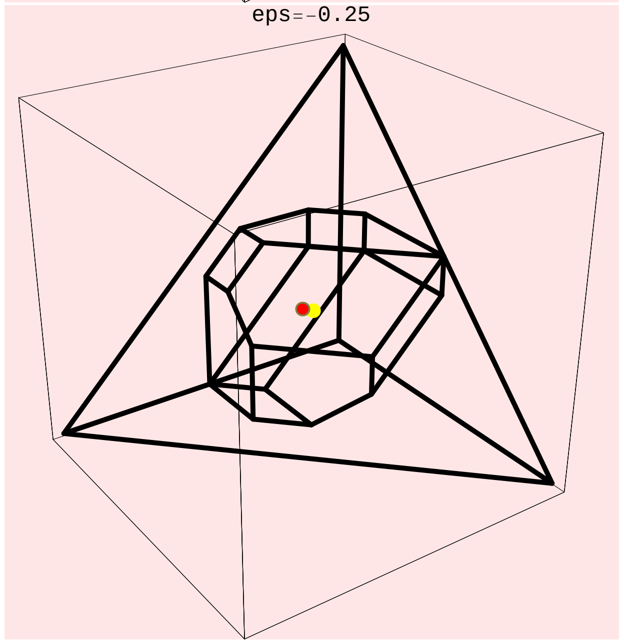

eps=-0.0416667


eps=-0.0833333

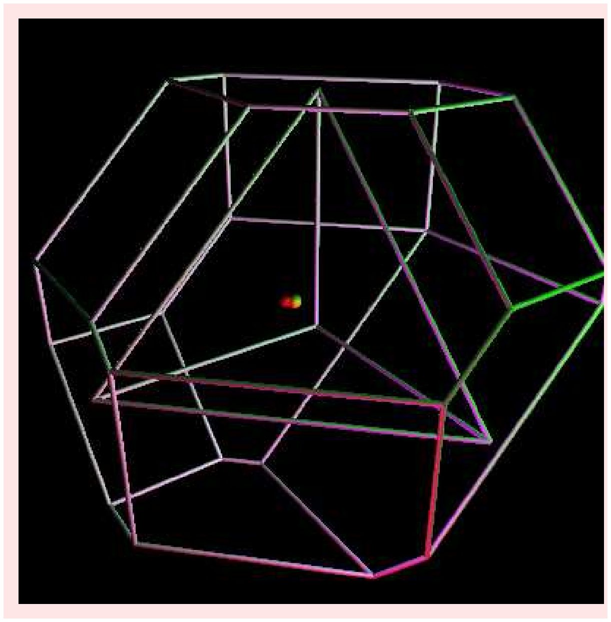

eps=-0.125







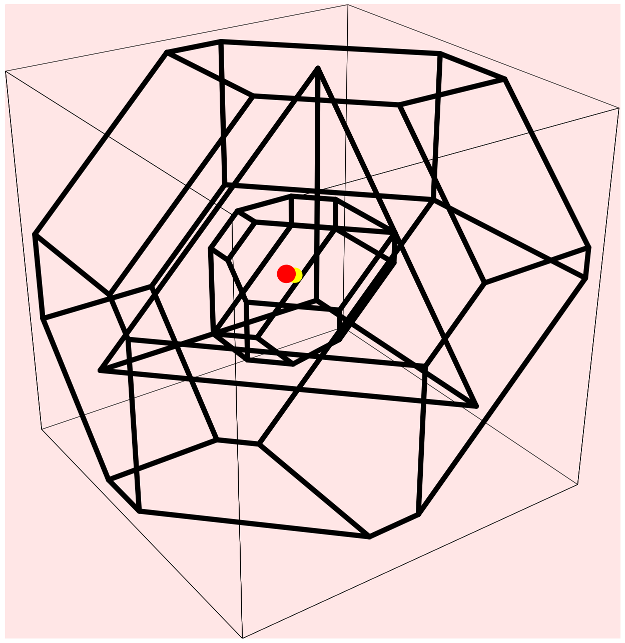
```
In[44]:= AbsoluteTiming[MvCorePlot3D[dir1, "ExpGameStrCore01", ExpGame, ShowStrongEpsCore → True,
           ViewKernelSol → False, ShowCore → False, KernelCoord → {ker1, ker12},
           ViewNucleolusSol → True, NucleolusCoord → nuc1, ViewShapleySol → True,
           ShapleyCoord → sh1, SaveAsEPS → False, SaveAsPOV → False]]
```



```
Out[44]= {6.72901 Second, Null}
```

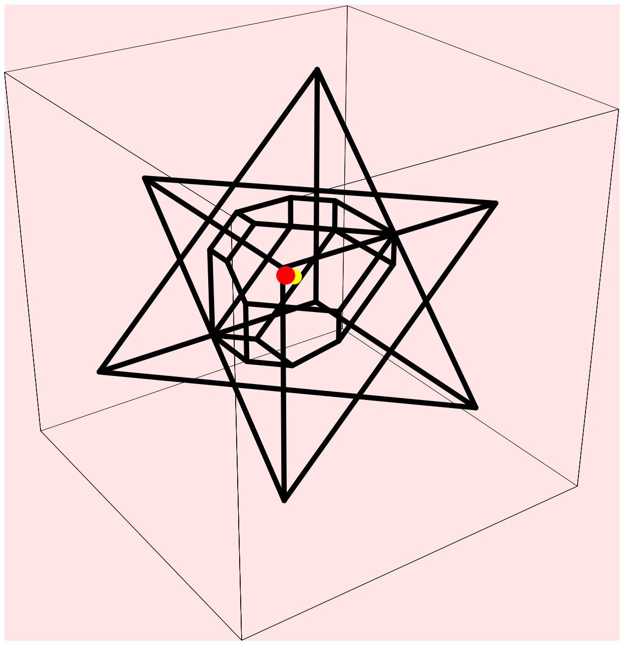
Instead of viewing some prominent solution concepts from cooperative game theory. We can also visualize some kernel catchers.

```
In[45]:= AbsoluteTiming[ShowKernelCatcher[dir1, "expgamecatcher", ExpGame, ShowCore → True,
           ShowStrongEpsCore → True, CriticalValue → {3}, ShowLowerSet → True, ShowUpperSet → False,
           ViewKernelSol → True, KernelCoord → ker1, ViewShapleySol → True, ShapleyCoord → sh]]
Reasonable set coincides with the Upper set!
The Lower Set coincides with the Imputation Set!
Plotting the Strong Epsilon Core, the Core, the Lower Set
 and the Imputation Set.
```



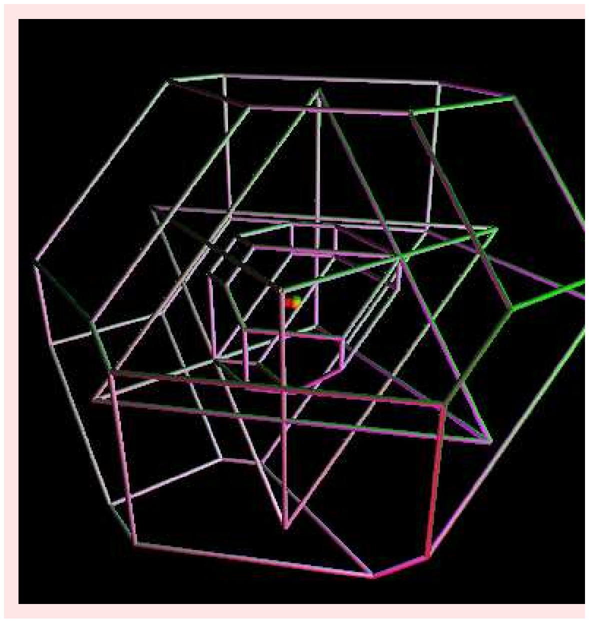
```
Out[45]= {1.8094 Second, Null}
```

```
In[46]:= AbsoluteTiming[ShowKernelCatcher[dir1, "expgamecatcher", ExpGame, ShowCore → True,
            ShowStrongEpsCore → False, CriticalValue → {3}, ShowLowerSet → True, ShowUpperSet → True,
            ViewKernelSol → True, KernelCoord → ker1, ViewShapleySol → True, ShapleyCoord → sh]]
Reasonable set coincides with the Upper set !
The Lower Set coincides with the Imputation Set !
Plotting the Core, the Lower, Upper Set
and the Imputation Set.
```



```
Out[46]= {1.71529 Second, Null}
```

```
In[47]:= AbsoluteTiming[MvKernelCatcher[dir1, "Expcatcher", ExpGame, ShowCore → True,
            ShowStrongEpsCore → True, ShowLowerSet → True, ShowUpperSet → True, ViewKernelSol → True,
            KernelCoord → ker1, ViewNucleolusSol → True, NucleolusCoord → nuc, ViewShapleySol → True,
            ShapleyCoord → sh]]
Reasonable set coincides with the Upper set !
The Lower Set coincides with the Imputation Set !
```



```
Out[47]= {9.73017 Second, Null}
```

3 References

M. Carter,Cooperative Games ,in Economic and Financial Modeling with MATHEMATICA®,editor Hal R.Varian, *Springer Publisher*,167-191,1993.

C. Chang and C-H. Lian, Some results on (Pre)Kernel catchers and the coincidence of the Kernel with the Prekernel, *International Game Theory Review*, Vol 4. No. 3, pp. 201-211, 2002.

C. Chang and T.S.H. Driessen, (Pre)Kernel Catchers for Cooperative Games, *OR Spectrum*, Vol. 17. pp. 23-26, 1995.

Y. Funaki, Upper and Lower Bounds of the Kernel and Nucleolus, *International Journal of Game Theory*, Vol. 15. pp. 121-129, 1986.

M. Maschler, B. Peleg and L.S. Shapley, Geometric Properties of Kernel, Nucleolus and related Concepts, *Mathematics of Operations Research*,Vol.4,Nov.1979,p.303-338.

H. Meinhardt, An LP approach to compute the pre-kernel for cooperative games, *Computers and Operation Research*, Vol 33/2 pp. 535-557,2006.