Newman C.E., Batteh J.J., Tiller M.:
Spark-Ignited-Engine Cycle Simulation in Modelica
2nd International Modelica Conference, Proceedings, pp. 133-142

Paper presented at the 2nd International Modelica Conference, March 18-19, 2002,
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Oberpfaffenhofen, Germany.

All papers of this workshop can be downloaded from
http://www.Modelica.org/Conference2002/papers.shtml

Program Committee:

- Martin Otter, Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Institut für Robotik und Mechatronik, Oberpfaffenhofen, Germany (chairman of the program committee).
- Hilding Elmqvist, Dynasim AB, Lund, Sweden.
- Peter Fritzson, PELAB, Department of Computer and Information Science, Linköping University, Sweden.

Local organizers:
Martin Otter, Astrid Jaschinski, Christian Schweiger, Erika Woeller, Johann Bals,
Deutsches Zentrum für Luft- und Raumfahrt e.V. (DLR), Institut für Robotik und Mechatronik, Oberpfaffenhofen, Germany

Spark-Ignited-Engine Cycle Simulation in Modelica

Charles Newman, John Batteh, and Michael Tiller
Ford Motor Company, USA

Abstract

This paper details the use of the Modelica modeling language for the cycle simulation of a spark-ignited engine. After a brief overview of the physical processes which must be modeled by a predictive cycle simulation model, this work emphasizes the two main challenges to the developer of such a model in Modelica: zone formation/destruction and calculation of realistic thermodynamic properties of the cylinder contents. The results illustrate that Modelica is capable of handling the complex physical models required by cycle simulation programs.

1 Introduction

Computer programs, which simulate the thermodynamic cycle of an internal combustion engine, have been developed over the last several decades both to assist in understanding the observed behavior of engines and to predict engine performance and efficiency as functions of engine design parameters (see [1], [2]). At Ford Motor Company the internally-developed General Engine Simulation (GESIM) program [3-6] has matured sufficiently that it can accurately predict the effects of intake and exhaust port design, combustion chamber geometry, and valve timing on combustion rate, fuel economy, and emissions for a spark-ignited engine.

Although very useful, GESIM has some significant limitations: it simulates only one cylinder of an engine running at constant angular velocity and reports the cycle averaged output torque. The result is essentially a simulation of a dynamometer test point for the engine. Currently, GESIM is written in procedural languages (FORTRAN and C), and its capabilities cannot readily be extended to include the transient multi-cylinder behavior required to simulate real engine operation in a vehicle. An effort is now under way to capture GESIM's physical models in Modelica [7, 8], thereby retaining its current capabilities, while removing the limitations on its applicability. Previous work [9, 10] proved the feasibility of this type of detailed powertrain modeling in Modelica. This paper, after a brief overview of the physical processes which must be modeled by a predictive cycle simulation model, focuses on the two main challenges to the developer of such a model in Modelica: zone formation/destruction and calculation of realistic thermodynamic properties of the cylinder contents.

2 Overview of Cycle Simulation Physics

The goal of a cycle-simulation program is to perform a thermodynamic analysis of the engine cylinder contents through each engine cycle, an overview of which is shown in Figure 1:

a) The mixture is prepared during the gas exchange period which extends from the time the exhaust valve opens (EVO) until the intake valve closes (IVC); during this period the burned gases are expelled, and a fresh mixture of fuel and air is inducted into the chamber. The mixture is then compressed until the piston reaches a position near top dead center (TDC).
b) In a spark-ignited engine, combustion is then initiated by the firing of the spark plug.
c) The mixture is then burned, raising the in-cylinder pressure and temperature considerably. In contrast to the process in a diesel engine, where combustion occurs throughout the chamber simultaneously, the mixture is consumed through the propagation of a well-defined flame front across the combustion chamber.
d) After the flame consumes all the combustible mixture, usually some time after TDC, the gas continues to expand, transferring energy to the piston as it continues its downward trajectory.

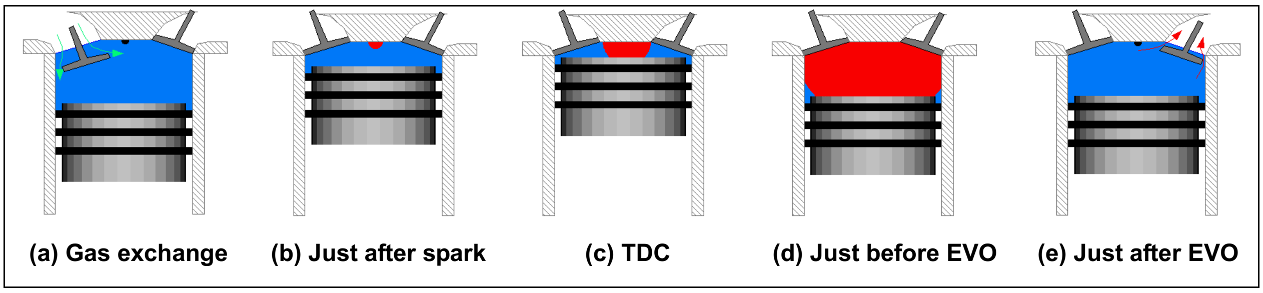


Figure 1. Modeling of cylinder contents at various points in engine cycle

e) When EVO is again reached, the process begins anew.

Successful modeling of the cycle requires capturing the essential physics of all the processes described above. In this paper, however, we concentrate on those processes critical to the thermodynamic analysis: combustion by a propagating flame and the computation of realistic thermodynamic properties of the gases comprising the in-cylinder mixture.

3 Combustion Modeling

The traditional approach taken to model the spark-ignition combustion process is suggested by Figure 1: we divide the cylinder contents into two (or more) thermodynamic zones, each with its own temperature and composition. Behind the flame is a zone comprised of only burned gases at high temperature (the burned zone). Ahead of the flame is the unburned zone, containing the remnants of the original mixture at a much lower temperature.

Each zone is regarded as a homogeneous mixture of *N* species (or pseudo-species), each modeled as an ideal gas. The zone must then satisfy the First Law of Thermodynamics,

$$\frac{dU}{dt} = \dot{Q} - P\frac{dV}{dt}, \tag{1}$$

the ideal gas law,

$$PV = MRT, \tag{2}$$

and the conservation of mass for each species

$$\frac{dm_i}{dt} = \dot{S}_i \tag{3}$$

where

$$\dot{Q} = \dot{H} + \dot{q}_w \tag{4}$$

$$M = \sum_i m_i \tag{5}$$

$$m_i = MX_i \tag{6}$$

$$U = Mu \tag{7}$$

$$R = \sum_i X_i R_i = \overline{R}\sum_i \frac{X_i}{\mu_i} \tag{8}$$

and

U	is the total internal energy of the zone
$\dot{Q}$	is the total energy flow into the zone
P	is the cylinder pressure
V	is the volume of the zone
$\dot{H}$	is the total flow of enthalpy entering the zone
$\dot{q}_w$	is the heat transferred from the chamber walls to the zone
M	is the total mass in the zone
R	is the overall (mass-specific) gas constant for the zone
T	is the temperature of the zone
m_i	is the mass of species *i* in the zone
X_i	is the mass fraction of species *i* in the zone
$\dot{S}_i$	is the total flow of species *i* of the mass flow entering the zone
u	is the specific internal energy of the zone
$u_i(P,T)$	is the specific internal energy of species *i* and is a known function of *P* and *T*
$\overline{R}$	is the universal gas constant
$\mu_i(P,T)$	is the average molecular weight of species *i* and is a known function of *P* and *T*

In addition to a set of equations (1)-(8) for each zone *z*, a constraint on the total volume must be added:

$$V_T = \sum_z V_z \tag{9}$$

where

V_T	is the total volume of the chamber

The task of the modeler is to complete the system of equations by supplying conditions to specify the flows appearing in equations (1)-(8) for all zones.

3.1 GESIM Implementation

GESIM makes two modifications to the above approach:

1. During mixture preparation (gas exchange and compression), the contents are treated as a single zone. No solution is sought for the quantities in the burned zone.
2. After combustion is initiated and as soon as the flame front makes contact with the surface of the chamber, the burned zone is further subdivided into an adiabatic core and a thermal boundary layer between the core and the wall.

GESIM implements the engine cycle of Figure 1 as follows:

a) The system to be solved consists of equations (1)-(9) for a single (unburned) zone, supplemented by the following relationships for the flows:

$$\dot{S}_{ui} = \sum_v \dot{m}_v \widetilde{X}_{vi} \tag{10}$$

$$\dot{H}_u = \sum_v \dot{m}_v \widetilde{h}_v \tag{11}$$

where

$\dot{m}_v$	is the mass flow into the zone through valve *v*
$\widetilde{h}_v$	is the specific enthalpy of the mass flow entering the zone through valve *v*
$\widetilde{X}_{vi}$	is the fraction of species *i* of the mass flow entering the zone through valve *v*

b) At spark, the solution is interrupted and a kernel (diameter ~ 1 mm) of burned gases is instantly created from the unburned mixture. This forms the initial state for the adiabatic zone. Equations (1)-(8) for the adiabatic zone are added to the system, and flows for both zones are now specified by

$$\dot{S}_{ui} = -\dot{m}_b X_{ui} \tag{12}$$

$$\dot{H}_u = -\dot{m}_b h_u \tag{13}$$

for the unburned zone and

$$\dot{S}_{bi} = \dot{m}_b B_i(\{X_u\}) \tag{14}$$

$$\dot{H}_b = \dot{m}_b h_u \tag{15}$$

for the burned zone where

$\dot{m}_b$	is the burn rate as calculated by the flame propagation model
$B_i(\{X_u\})$	is the fraction of species remaining after a mixture of composition X_{ui} is burned.
h_u	is the specific enthalpy of the unburned zone

c) When the flame contacts the wall, the solution is interrupted and a thin boundary layer is initialized and the system of equations altered in a manner similar to b) above. The details are omitted here.

d) At EVO, the solution is again interrupted. All zones are mixed together instantly to form a single zone, which represents the initial state for the unburned zone for the next cycle.

e) The set of equations is reduced to those of a) above and the solution is resumed.

3.2 Modelica Implementation

As GESIM is an in-house product written in FORTRAN, it has complete control over the solution method- it can interrupt the solution at will to expand or shrink the system of equations, reinitialize, and resume. In Modelica, however, where the number of equations is fixed, we adopt two modifications to GESIM's approach:

1. The burned zone (*i.e.* the adiabatic zone and boundary layer) exists throughout the simulation. Each zone satisfies equations (1)-(8), and equation (9) is imposed on the volumes. During the mixture preparation period, when the burned zone does not exist in GESIM, we require that it have a small mass (less than the initial spark kernel) and have temperature and composition equal to that of the unburned zone. The solution for the two zones degenerates to an equivalent single-zone simulation during this portion of the cycle.
2. GESIM effects the transitions between 1-zone and multi-zone behavior essentially by simulating impulses. In the absence of a stable and mature impulse capability in Modelica, we choose to perform these transitions over a finite, but short time.

Since all zones are always mathematically active, enough conditions must be supplied to specify all the flows. While our model includes all three zones, in this paper we discuss only the aspects of modeling two zones in order to illustrate the approach. Adding the boundary layer is a relatively straightforward extension of the technique.

During mixture preparation, the "burned" zone is just a dummy placeholder, containing a small amount of mass in its unburned state, which will be the first mass to be burned in forming the initial kernel after spark. We require that both zones have identical temperature and composition. Since (5) and (6) imply that $\sum_i X_i = 1$, only N-1 components of the composition vector can be independently constrained. Hence, our condition can be expressed as

$$\Delta T = T_u - T_b = 0 \tag{16}$$

$$\Delta X_i = X_{ui} - X_{bi} = 0,\quad 1 \le i \le N-1 \tag{17}$$

$$\Delta V = V_b - V_K = 0 \tag{18}$$

where

V_K	is a volume small compared to that of the initial spark kernel

Denoting the time of EVO as t_0, we achieve the transition from combustion to the above conditions by mixing the contents of the two zones over a short time $\tau_s \sim 100\mu s$; *i.e.*, for $t_0 \le t \le t_F = t_0 + \tau_s$:

$$\Delta T = \Delta T_0 \sigma(t) \tag{19}$$

$$\Delta X_i = \Delta X_{i0} \sigma(t),\quad 1 \le i \le N-1 \tag{20}$$

$$\Delta V = \Delta V_0 \sigma(t) \tag{21}$$

where the subscript 0 denotes the value of a quantity at EVO and

$$\sigma(t) = (\frac{t - t_F}{\tau_s})^2 . \tag{22}$$

Referring again to Figure 1, the Modelica implementation of the cycle is as follows:

a) During mixture preparation, a dummy burned zone (shown in black) exists. Except for the transition time at EVO, its volume is V_K. Otherwise it is indistinguishable from the main unburned zone. To specify the flows, we first introduce modified forms of (10)-(11):

$$\dot{S}_{ui} + \dot{S}_{bi} = \sum_v \dot{m}_v \widetilde{X}_{vi} \tag{23}$$

$$\dot{H}_u + \dot{H}_b = \sum_v \dot{m}_v \widetilde{h}_v \tag{24}$$

The constraints (16)-(18) or (19)-(21) are sufficient to complete the specification for the transition at EVO or the main portion of mixture preparation, respectively. Differentiating both of the above sets of equations, we can combine them into a single set. Between EVO and spark, then

$$\frac{d\Delta T}{dt} = \Delta T_0 \lambda(t) \tag{25}$$

$$\frac{d\Delta X_i}{dt} = \Delta X_{i0} \lambda(t),\quad 1 \le i \le N-1 \tag{26}$$

$$\frac{d\Delta V}{dt} = \Delta V_0 \lambda(t) \tag{27}$$

where

$$\lambda(t) = \frac{2}{\tau_s} \min(0, \frac{t - t_F}{\tau_s}) . \tag{28}$$

b) The transition at spark from mixture preparation to combustion is accomplished in two steps.

1. Denoting the time of spark as t_I, we first burn the contents of the "burned" zone over a time $\tau_1 \sim 1\mu s$; *i.e.*, for $t_I \le t \le t_M = t_I + \tau_1$, the flows are specified by

$$\dot{S}_{ui} = 0 \tag{29}$$

$$\dot{H}_u = 0_u \tag{30}$$

$$\dot{S}_{bi} = \frac{M_{bI}}{\tau_1}[B_i(\{X_I\}) - X_{Ii}] \tag{31}$$

$$\dot{H}_b = 0 \tag{32}$$

where the subscript I denotes a value at ignition.

2. At $t = t_M$, two-zone combustion begins. Equations (12)-(15) are used unchanged, just as in GESIM, with the burn rate $\dot{m}_b$ initially set high enough to assure that, by $t = t_M + 1\mu s$, a burned zone volume will be achieved equal to that of GESIM's initial spark kernel. As soon as the required volume is attained, the flame propagation model is used to calculate the burn rate.

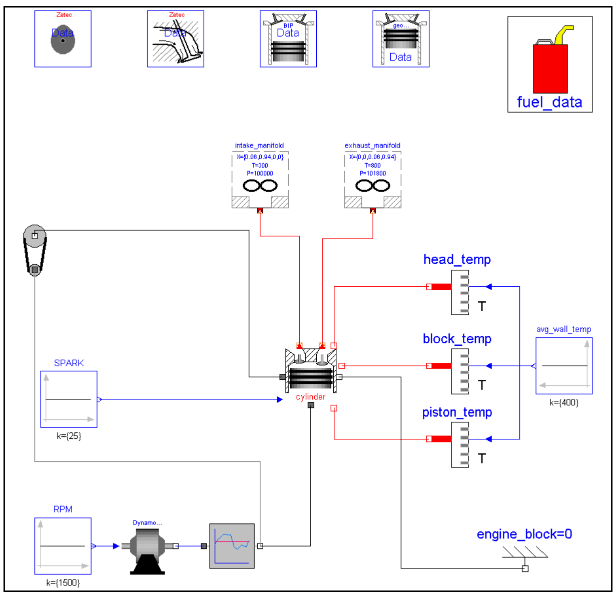
c) The main phase of combustion is identical to that of GESIM.

d) At EVO, expansion ends. The current time is assigned to t_0, and we change over to the mixture preparation phase.

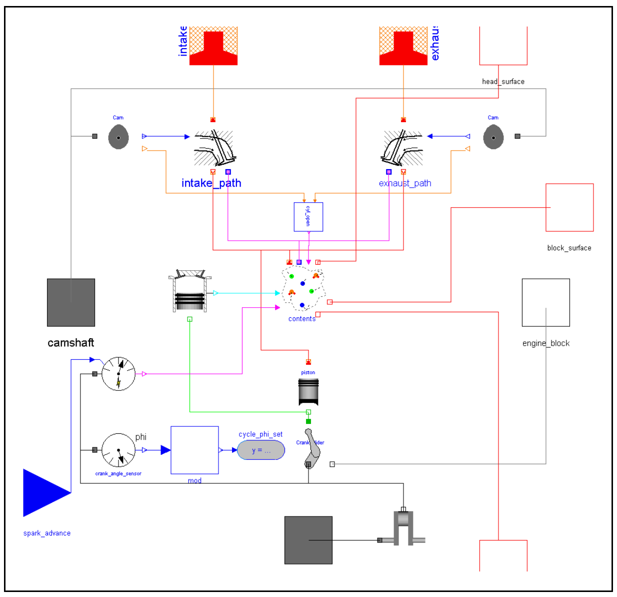
e) The next cycle begins.

Figure 2 shows schematics for successive levels of the instance hierarchy in the Modelica implementation of a single-cylinder version of this model. The engine itself is shown in Figure 2(a). Its `cylinder` component appears in Figure 2(b); it has been designed to facilitate construction of multi-cylinder configurations through its replication. The `contents` component of the cylinder, shown in Figure 2(c), models the thermodynamics of all gases residing in the cylinder. Finally, in Figure 2(d) we see the `combustion` component, the main focus here.

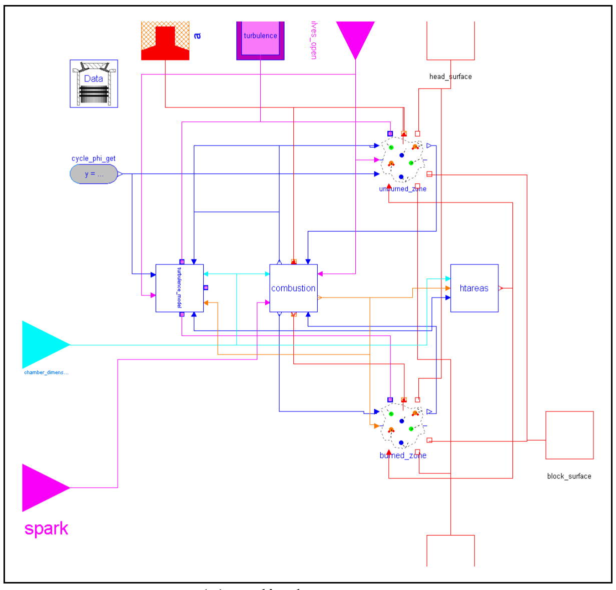
The `combustion` component controls the creation, evolution, and destruction of the burned zone in the manner discussed above by coordinating the activities of a parallel configuration of three subcomponents: `mix_zones` to mix burned and unburned together at EVO by specifying the flows according to a) above; `kernel_burn` to create the initial spark kernel as described in b) above; `flameadv` to grow the burned zone according b) and c). Each of these components supplies non-vanishing contributions to the total flow only during the portion of the cycle that it is meant to control. Figures 6-9, included at the end of the paper, contain code fragments that provide some insights into how the models described in this section have been implemented.



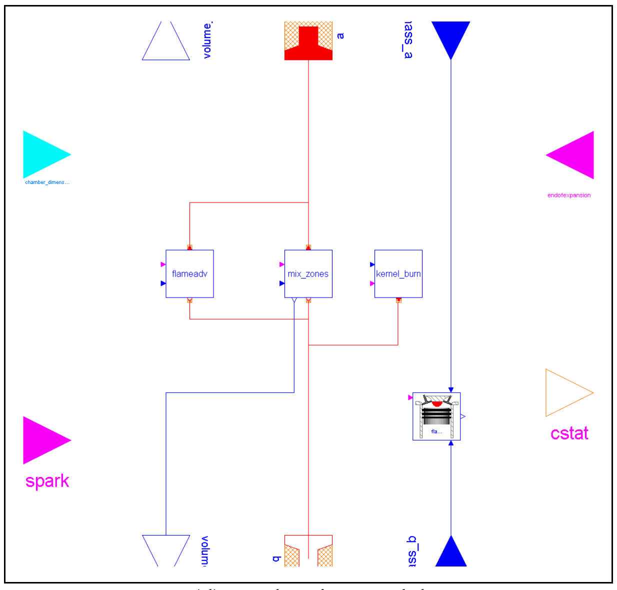
(a) single cylinder engine



(b) one cylinder



(c) cylinder contents



(d) combustion model

Figure 2. Schematic representations of the engine model hierarchy

4 Thermodynamic Properties

4.1 The "`MediumModel`" Idiom

Different engine simulation applications require different levels of detail. One of the important determinations to be made is what level of detail is required in computing the thermodynamic properties of the cylinder contents. In some cases, we can treat the medium flowing through the engine as simply air but in other cases we might need to allow for changes in composition of the gas that would require tracking several chemical species.

At first glance, it would appear that different component models (*e.g.* valves, control volumes) would be required for each of the possible media. But if we look carefully at the issue, we find that the properties of the selected medium are orthogonal to the equations of the various thermodynamic processes. In other words, if the models are formulated correctly, the choice of media can be made independently of the components used to model the engine cycle.

In practice, this is achieved by using what we refer to as the `MediumModel` idiom. The basic idea behind this idiom is to define a `partial package` that describes the interfaces of the various models, connectors, *etc.* that will be required to implement all of our component models. However, no implementation is provided by this `partial package`. This is essentially a Modelica adaptation of the "Kit" or "Abstract Factory" pattern found in [13]. In the same way that a "Kit" might be used as a means of instantiating compatible GUI toolkit components such as scrollbars, menus, *etc.*, the `MediumModel` is used to instantiate consistent sets of property models, connectors, *etc.*

The complete definition of the `MediumModel` package definition is too lengthy to include here, but it consists mainly of three things. First, it contains a `partial model` definition that defines the interface for computing medium properties. Second, it contains `partial connector` definitions that include the appropriate number of chemical species flowing between components. Finally, it contains several `partial function` definitions for computing useful quantities (*e.g.* air fuel ratio) using medium composition information.

4.2 Property Calculations

As discussed previously, the conservation of energy for the various combustion zones in the cylinder is at the heart of the cycle simulation tool. In addition, a specific medium model is needed to determine the thermodynamic properties (*e.g.* specific enthalpy, h, and specific internal energy, u) of the cylinder contents used in Eqs. (4), (7).

In simple combustion simulations the cylinder contents can be treated as a single ideal gas. Constructing a medium model for a single ideal gas is relatively easy. Since the thermodynamic properties vary as a function of temperature only, they can be calculated from a look-up table or a polynomial regression of tabulated data. However, for detailed combustion systems the medium is assumed to be a reacting **mixture** of ideal gases (*e.g.* the fuel vapor, air, and combustion products). Therefore, in order to compute the contribution of each species to the mixture property we must first determine the relative amounts of the various species in the mixture. This calculation requires the solution of the nonlinear system of equations that define chemical equilibrium for the mixture.

The steps required for the property calculation are detailed in [11] and can be summarized as follows:

1. Solve the nonlinear system of equations that defines chemical equilibrium for the combustion mixture to yield the mixture composition
2. Calculate the contribution of each species to the mixture property
3. Calculate the mixture property from the individual species contributions and the mixture composition

While the calculations in steps 2 and 3 above are simple evaluations, the nonlinear solution of the chemical equilibrium problem is certainly nontrivial. In the GESIM property models the combustion products are comprised of twenty-one species; thus, obtaining the mixture composition requires the solution of a set of twenty-two nonlinear equations for chemical equilibrium. Furthermore, recall that h and u are functions of P, T, and ϕ (the equivalence ratio of the combustion products). Therefore, the property calculations, including the determination of the equilibrium composition, must be computed for each thermodynamic zone in the engine model whenever there is a change in the pressure, temperature, and/or composition of any of the thermodynamic zones.

While this method for calculating the mixture properties could be implemented directly in Modelica, it would require the cycle simulation tool to repeatedly compute the solution of the chemical equilibrium problem, a formulation that has several drawbacks. First, the repeated solution of the nonlinear equations used to determine the equilibrium chemistry is computationally demanding when compared to the other behavior equations involved, a formulation which would result in slower simulation times. In addition, the chemical systems introduce other issues such as robustness of the nonlinear solution method and scaling of the chemical concentrations. So, rather than calculate the needed properties on-demand during the simulations, an alternative approach is to pre-compute the properties throughout the expected domain of operation and simply interpolate as needed during the simulations. The `ModelicaAdditions` package contains a `Tables` package that includes models for linear interpolation in one and two dimensions, `CombiTable1D` and `CombiTable2D` respectively. However, the mixture properties are functions of three variables (P, T, and ϕ). Even tri-linear interpolation would not suffice as continuity of the mixture properties and gradients could not be insured. It can be shown [11] that this continuity in gradients is important for fast and accurate simulation. Furthermore, maintaining continuous gradients allows for index reduction in Modelica and would certainly be of benefit to the numerical integration schemes in Dymola [12].

As a result, higher order interpolation schemes are required to provide the desired continuity. They involve the construction and evaluation of polynomials to yield the interpolated values and are more difficult to implement since more information is needed about the function other than simply its value at each grid point (*i.e.* the derivatives of the function, additional function values at adjoining cubes, *etc.*).

Though not detailed in this work, a flexible modular scheme has been developed to automatically formulate the chemical equilibrium problem and solve for the equilibrium chemistry and mixture properties over a wide range of engine operating conditions (P, T, and ϕ). The remainder of this section discusses the implementation of a higher order interpolation scheme in Modelica with the assumption that a file has been created that contains all the necessary data to perform the interpolation.

4.3 Hermite Interpolation

Based on the requirements detailed in the previous section, the interpolation scheme must be three-dimensional and provide continuity of the interpolated function value and its derivative. One scheme that satisfies those criteria is Hermite interpolation [14]. The Hermite interpolating function for a generic property p is defined in standard tensor notation as follows:

$$p(u,v,w) = F_l(u)F_m(v)F_n(w)\mathbf{b}_{\mathbf{lmn}} \qquad (33)$$

where the following vectors define the cubic blending functions:

$$\begin{aligned} F_1(u) &= 2u^3 - 3u^2 + 1 \\ F_2(u) &= -2u^3 + 3u^2 \\ F_3(u) &= u^3 - 2u^2 + u \\ F_4(u) &= u^3 - u^2 \end{aligned} \qquad (34)$$

and similarly for $F_m(v)$ and $F_n(w)$. The blending functions clearly show the cubic nature of the interpolating polynomial and are evaluated based on the point within the cube at which the interpolated value is sought, denoted by the star in Figure 3.

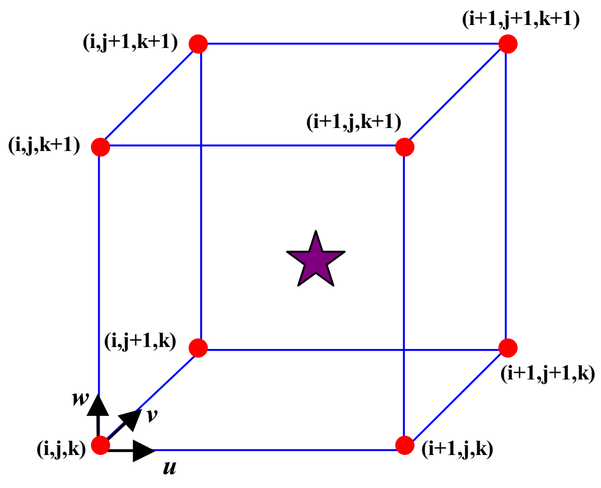


Figure 3. Hermite interpolation cube

The tensor $\mathbf{b}_{\mathbf{lmn}}$ is comprised of externally-provided data about the function p and its derivatives, data that is required at the eight cube vertices shown and labeled in Figure 3. The data consists of eight pieces of information at each of the eight vertices, a total of 64 pieces of information for a single cube: the function value, three tangent vectors, three twist vectors, and a vector defined by the third-order mixed partial derivative of the function. See [14] for a complete description of the Hermite interpolation scheme and data required.

Clearly a significant amount of data is required for the interpolation of the thermodynamic properties h and u. With some symbolic manipulation of the property equations, it is possible to derive all the necessary function and derivative data analytically without resorting to numerical differentiation. This data is available to Modelica in the form of a Matlab .mat file. A typical 30 x 45 x 45 data file is approximately 7.6 MB.

4.4 Modelica Implementation

Once we have decided on an appropriate interpolation scheme and collected all the property data required, the next step is to implement the interpolation scheme so that it can be used from within our Modelica models. In our implementation, the interpolation and gradient calculations (associated with the interpolation function via the `derivative` annotation) were written in "C". These functions are then called as `external` functions by native Modelica functions.

While the steps required are straightforward, there are several implementation details worth discussing. For example, in order to perform the interpolation, the property data must be loaded and made available to the "C" language routines. Rather than load the data (which is quite voluminous) into Modelica arrays and pass it as an argument to the various functions, we chose instead to load the data into memory and simply refer to it using an integer identification number. As a result, the only data passed around in the Modelica models is the unique ID number that identifies where the data can be found in memory. In the future, the interpolation routines will be upgraded to use the newly adopted `ExternalObject` class in Modelica 2.0 [8] that was introduced to provide more direct support for these kinds of applications.

Another issue with the interpolation routines is to improve performance by implementing some form of caching mechanism. There are two reasons to implement a cache mechanism. First, the simulation tool may not entirely optimize away redundant function calls (*i.e.* calls with the same arguments and therefore the same results). In such cases, a cache can be used to store previous results and avoid expensive recalculations. Another reason to implement a cache is to allow for common calculation to be shared among the various interpolation-related functions. For example, calculating the gradient of the interpolating function requires much of the same data as the function evaluation itself. These common quantities can also be stored in a cache and reused across calculations.

Once we have implemented the interpolation routines, we can move on to implementing a medium model that utilizes these interpolation routines. This calculation involves two different interpolations. First, the properties of the gaseous air-fuel mixture (which is treated as a non-reacting mixture of ideal gases) can be computed via interpolation in temperature. Then, the properties of the reacting combustion products are computed using interpolation in pressure, temperature and equivalence ratio. These two sets of properties are then combined to form a single set of properties for the entire air, fuel and combustion products mixture.

Although we implemented our own interpolation routines for this purpose, we will work toward incorporating similar functionality into the Modelica Standard Library so that the routines can be more fully optimized and so that future users will be able to simply reuse what is in the library rather than having to create their own.

5 Cycle Simulation Results

The single-cylinder Modelica model was run for 1 second of simulation time at 1500 rpm at slightly lean conditions. By the end of the tenth engine cycle, approximately 0.8 seconds, the model has effectively converged to "steady-state"; *i.e.*, each cycle is a repeat of its predecessor. Some results for the tenth cycle are shown in Figures 4-5.

Figure 4 plots the temperature of all three zones. During mixture preparation the temperatures are equal, as is required. At spark, they separate, with the adiabatic temperature exceeding 2600 K, the unburned zone peaking at around 900 K, and the boundary layer achieving a value in between. These temperatures agree well with those computed in GESIM and are typical of those encountered in spark-ignited engines. At EVO, when the zones are remixed, all temperatures again quickly collapse to a single value, as expected.

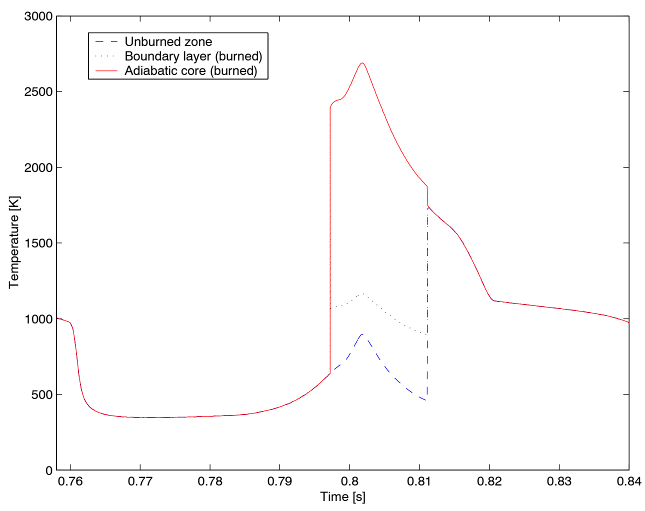


Figure 4. Zone temperatures (converged cycle)

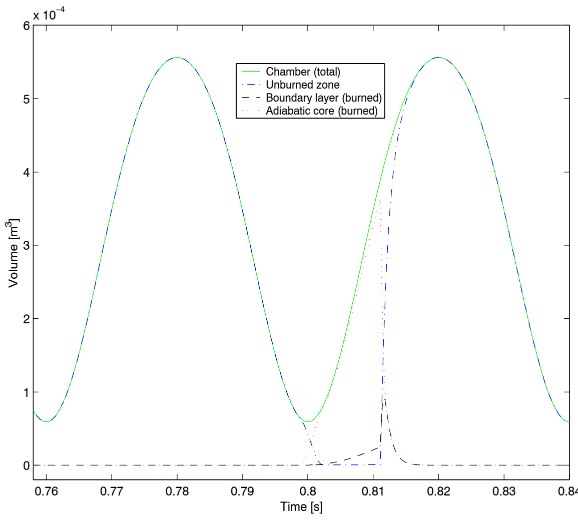


Figure 5. Zone volumes (converged cycle)

The volumes of the three zones, along with the total chamber volume, are plotted in Figure 5. Since some of the zones during parts of the cycle are artifices of our modeling approach, we cannot use measurements or GESIM to gauge their accuracy. However, they do behave as anticipated. During mixture preparation, when the adiabatic zone and the boundary layer do not appear in GESIM, their volumes are indeed insignificant. When combustion begins those two zones grow rapidly at the expense of the unburned zone, eventually reducing the size of the latter to insignificance at the end of combustion. At EVO, when the zones are remixed, all zones quickly revert to their values for mixture preparation.

6 Conclusions

This paper outlines the handling of zone formation/destruction and calculation of realistic thermodynamic properties of the cylinder contents in Modelica for engine cycle simulation. The results illustrate that Dymola and Modelica are capable of handing the complex physical models required for predictive cycle simulation. Furthermore, the techniques used provide illustrative examples for the handling of similar behavior in different applications.

Acknowledgements

The authors would like to acknowledge Dr. Anne Marsan at Ford Research Laboratories for the helpful discussions on geometry and interpolation. In addition, we would also like to acknowledge the support provided by Dynasim AB during the development of these models.

References

1. Heywood, J.B., 1988, *Internal Combustion Engine Fundamentals*. McGraw-Hill.
2. Tiller, M. M., 2001, *Introduction to Physical Modeling with Modelica*. Kluwer.
3. Borgnakke, C., *et al.*, 1980, "A Model for the Instantaneous Heat Transfer and Turbulence in a Spark Ignition Engine," SAE-80-0287, Society of Automotive Engineers.
4. Newman, C.E., *et al.*, 1989, "The Effects of Load Control with Port Throttling at Idle---Measurements and Analyses", SAE-89-0679, Society of Automotive Engineers.
5. Brehob, D. D., and C. E. Newman, 1992, "Monte Carlo Simulation of Cycle by Cycle Variability," SAE-92-2165, Society of Automotive Engineers.
6. Miller, R., *et al.*, 1998, "Comparison of Analytically and Experimentally Obtained Residual Fractions and NO_x Emissions in Spark-Ignited Engines", SAE-98-2562, Society of Automotive Engineers.
7. Modelica Association, 2000, "Modelica Language Specifications (Version 1.4)", www.modelica.org.
8. Modelica Association, 2002, "Modelica Language Specifications (Version 2.0)", www.modelica.org.
9. Tiller, M.M., *et al.*, 2000, "Detailed Vehicle Powertrain Modeling in Modelica", Modelica Workshop 2000 Proceedings, pp. 169-178.
10. Bowles, P., *et al.*, 2001, ″Feasibility of Detailed Vehicle Modeling″, SAE-2001-01-0334, Society of Automotive Engineers.
11. Olikara, C. and Borman, G. L., 1975, ″A Computer Program for Calculating Properties of Equilibrium Combustion Products with Some Applications to I.C. Engines″, SAE-75-0468, Society of Automotive Engineers.
12. Dymola. Dynasim AB, Lund, Sweden, www.dynasim.se.
13. Gamma, E., 1995, *Design Patterns*. Addison-Wesley.
14. Mortenson, M.E., 1985, *Geometric Modeling*. John Wiley and Sons.

Code Fragments

This section contains code fragments to illustrate some of the points raised during discussion of zone formation and destruction.

```
  constant Integer mixprep=1;
  constant Integer ignstep1=2;
  constant Integer ignstep2=3;
  constant Integer propagating=4;
  constant Integer expanding=5;
  parameter Modelica.SIunits.Time tau=1e-4;
  parameter Modelica.SIunits.Time ign_delta=1e-6;
  Integer status;
  discrete Modelica.SIunits.MassFlowRate
    ignition_rate "Combustion rate during ignition";
  discrete Modelica.SIunits.Time endstep(start=tau);
equation
  mix_zones.mix.signal[1] = status == mixprep;
  mix_zones.tfinal.signal[1] = endstep;
  kernel_burn.burn.signal[1] = status == ignstep1;
  kernel_burn.burn_rate.signal[1] = burn_rate;
  flameadv.burn.signal[1] = status == ignstep2 or
                            status == propagating;
  flameadv.burn_rate.signal[1] = burn_rate;
.
.
.
  if status == ignstep1 or status == ignstep2 then
    burn_rate = pre(ignition_rate);
  elseif status == propagating then
    // post_ignition_rate is computed by flame
    // propagation model
    burn_rate = post_ignition_rate;
  else
    burn_rate = 0.0;
  end if;

algorithm
  when status == mixprep and spark.signal[1] then
    status := ignstep1;
    endstep := time + ign_delta;
    kernel_mass := pre(burned_mass);
    ignition_rate := kernel_mass/ign_delta;
  end when;

  when status == ignstep1 and time > endstep then
    status := ignstep2;
    endstep := time + ign_delta;
    ignition_rate := pre(burned_mass)*
           (2.0*initial_kernel_size/pre(burnedV) -
           1.0)/ign_delta;
  end when;

  when status == ignstep2 and
      (time > endstep or
       burnedV/initial_kernel_size > 1.0) then
    status := propagating;
  end when;

  when status <> mixprep and
      endofexpansion.signal[1] then
    status := mixprep;
  end when;

  when status == mixprep then
    endstep := time + tau;
  end when;
```

Figure 6. Excerpt from the `combustion` model

```
  parameter Modelica.SIunits.Volume Vbtarget
    "Unburned kernel size";
  constant Integer nindep=MediumModel.nspecies - 1
    "Number of independent species fractions";
  discrete Modelica.SIunits.Time endstep;
  Modelica.SIunits.MassFraction dX[nindep];
  Modelica.SIunits.Temperature dT=a.T - b.T;
  Real dV=1.0 - Vbnorm;
  Real Vbnorm(start=5, fixed=true);
  discrete Modelica.SIunits.Temperature deltaT;
  Modelica.SIunits.MassFraction deltaX[nindep];
  discrete Real deltaV
  discrete Real r;
  Real rate_expr;
.
.
.
equation
  // component a refers to the unburned zone, b to
  // the burned zone
  a.P = b.P;
  a.q + b.q = 0.0;
  a.mdot = -b.mdot;
  // volume_b.signal[1] is the volume of the
  // burned zone
  Vbnorm = volume_b.signal[1]/Vbtarget;
  rate_expr = 2.0*r^2*min(0.0, time - pre(endstep));
  dX = a.X[1:nindep] - b.X[1:nindep];
  if mix.signal[1] then // true at EVO
    der(dV) = pre(deltaV)*rate_expr;
    der(dX) = pre(deltaX)*rate_expr;
    der(dT) = pre(deltaT)*rate_expr;
  else
    a.q = 0.0;
    a.mdot = zeros(MediumModel.nspecies);
  end if;

algorithm
  when mix.signal[1] then
    deltaX := dX;
    deltaT := dT;
    deltaV := dV;
    endstep := tfinal.signal[1];
    r := 1.0/(endstep - time);
  end when;
```

Figure 7. Excerpt from the `mix_zones` model

```
equation
  // Component medium is the burned zone
  if burn.signal[1] then
    // MediumModel.BurnMixture computes the
    // composition after a given mixture is burned
    medium.mdot = burn_rate.signal[1]*
        (Xbase - MediumModel.BurnMixture(Xbase);
  else
    medium.mdot = zeros(MediumModel.nspecies);
  end if;
  medium.q = 0.0;

algorithm
  when burn.signal[1] then
    Xbase := medium.X;
  end when;
```

Figure 8. Excerpt from the `kernel_burn` model

```
equation
  a.q + b.q = 0;
  if burn.signal[1] then
    a.q = burn_rate.signal[1]*a.h;
    a.mdot = burn_rate.signal[1]*a.X;
    // MediumModel.BurnMixture computes the
    // composition after a given mixture is burned
    b.mdot = -burn_rate.signal[1]*
                 MediumModel.BurnMixture(a.X);
  else
    b.q = 0;
    a.mdot = zeros(MediumModel.nspecies);
    b.mdot = zeros(MediumModel.nspecies);
  end if;
```

Figure 9. Excerpt from the `flameadv` model